User's Manual

# Renesas Flexible Software Package (FSP) V6.2.0

## User's Manual

## Renesas RA Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Revision 5.20 Oct.15.2025

# Table of Contents


**Chapter 1 Introduction** . . . 14
1.1 Overview . . . 14
1.2 Using this Manual . . . 14
1.3 Documentation Standard . . . 14
1.4 Introduction to FSP . . . 14
1.4.1 Purpose . . . 15
1.4.2 Quality . . . 15
1.4.3 Ease of Use . . . 15
1.4.4 Scalability . . . 15
1.4.5 Build Time Configurations . . . 15
1.4.6 e² studio IDE . . . 15
**Chapter 2 Reference Materials** . . . 16
2.1 Using Registers Directly . . . 16
2.1.1 Overview . . . 16
2.1.2 What's in an . . . 16
2.1.3 Where are iodefine files located? . . . 16
2.1.4 Using the register definitions . . . 17
2.1.5 Tips for writing hardware drivers . . . 18
2.2 FSP v4.0.0 FreeRTOS Stack Migration Guide . . . 19
2.2.1 Overview . . . 19
2.2.2 Stack Migration Steps . . . 19
2.2.3 List of Code Change Highlights . . . 21
2.3 Cortex-M85 Caches . . . 22
2.3.1 Changes . . . 22
2.3.2 Overview . . . 22
2.3.3 CM85 Cache Features . . . 24
2.3.4 RA8 Cache Background Information . . . 25
2.3.5 Cache Maintenance . . . 26
2.3.6 Typical Cache Maintenance Scenarios . . . 26
2.3.7 Cache Functions and Macros . . . 29
2.3.8 Cache Details . . . 31
2.3.9 Other Information . . . 33
2.3.9.1 Cache ECC with FSP . . . 36
2.3.10 References . . . 37
2.4 Using Ocrypto . . . 39
2.4.1 Overview . . . 39
2.4.2 Limitations . . . 39
2.4.3 RA FSP Licensed Algorithms . . . 39
2.4.4 API Examples . . . 39
2.4.5 Additional Information . . . 40
2.4.6 Ocrypto - Side Channel Safety Statement . . . 40
2.4.7 Ocrypto - Function Overview . . . 43
2.5 Linker Script Generation Guide . . . 45
2.5.1 FSP Linker Scripts . . . 45
2.5.1.1 Why we changed . . . 45
2.5.1.2 New section model (high level) . . . 46
2.5.1.3 Interacting with the new flow . . . 47
2.5.1.4 Migration guide (step?by?step) . . . 49
2.5.1.5 Legacy (gcc, llvm) ? New **output** section translation . . . 49
2.5.1.6 Behavior differences worth noting . . . 50
2.5.1.7 Examples . . . 50

2.5.2 FAQ . . . 51
2.5.3 Summary . . . 51
2.5.4 Appendix: Linker Input Sections . . . 51
2.5.4.1 Memory Class Dictionaries . . . 52
2.5.4.2 RAM-Resident Sections . . . 52
2.5.4.3 ROM-Resident Sections . . . 53
2.5.4.4 Summary . . . 53
**Chapter 3 Starting Development** . . . 55
3.1 Starting Development Introduction . . . 55
3.2 e² studio User Guide . . . 56
3.2.1 What is e² studio? . . . 56
3.2.2 e² studio Prerequisites . . . 58
3.2.2.1 Obtaining an RA MCU Kit . . . 58
3.2.2.2 PC Requirements . . . 58
3.2.2.3 Installing e² studio, platform installer and the FSP package . . . 58
3.2.2.4 Choosing a Toolchain . . . 58
3.2.2.5 Licensing . . . 59
3.2.3 What is a Project? . . . 59
3.2.4 Creating a Project . . . 61
3.2.4.1 Creating a New Project . . . 61
3.2.4.2 Selecting a Board and Toolchain . . . 62
3.2.4.3 Selecting Flat or Arm TrustZone Project . . . 63
3.2.4.4 Selecting a Project Template . . . 64
3.2.5 Configuring a Project . . . 67
3.2.5.1 Summary Tab . . . 67
3.2.5.2 Configuring the BSP . . . 68
3.2.5.3 Configuring Clocks . . . 69
3.2.5.4 Configuring Pins . . . 70
3.2.5.5 Configuring Interrupts from the Stacks Tab . . . 72
3.2.5.6 Viewing Event Links . . . 75
3.2.6 Adding Threads and Drivers . . . 75
3.2.6.1 Adding and Configuring HAL Drivers . . . 76
3.2.6.2 Adding Drivers to a Thread and Configuring the Drivers . . . 78
3.2.6.3 Configuring Threads . . . 82
3.2.7 Reviewing and Adding Components . . . 83
3.2.8 Writing the Application . . . 84
3.2.8.1 Coding Features . . . 84
3.2.8.2 HAL Modules in FSP: A Practical Description . . . 90
3.2.8.3 RTOS-Independent Applications . . . 91
3.2.8.4 RTOS Applications . . . 92
3.2.8.5 Additional Resources for Application Development . . . 94
3.2.9 Debugging the Project . . . 95
3.2.10 Modifying Toolchain Settings . . . 95
3.2.11 Creating RA project with Arm Compiler 6 in e² studio . . . 97
3.2.12 Importing an Existing Project into e² studio . . . 100
3.2.13 Using Semihosting in a Project . . . 104
3.3 Tutorial: Your First RA MCU Project - Blinky . . . 105
3.3.1 Tutorial Blinky . . . 105
3.3.2 What Does Blinky Do? . . . 105
3.3.3 Prerequisites . . . 105
3.3.4 Create a New Project for Blinky . . . 105
3.3.4.1 Details about the Blinky Configuration . . . 108
3.3.4.2 Configuring the Blinky Clocks . . . 109
3.3.4.3 Configuring the Blinky Pins . . . 109
3.3.4.4 Configuring the Parameters for Blinky Components . . . 109

3.3.4.5 Where is main()? . . . 109
3.3.4.6 Blinky Example Code . . . 109
3.3.5 Build the Blinky Project . . . 109
3.3.6 Debug the Blinky Project . . . 110
3.3.6.1 Debug prerequisites . . . 110
3.3.6.2 Debug steps . . . 111
3.3.6.3 Details about the Debug Process . . . 112
3.3.7 Run the Blinky Project . . . 113
3.4 Tutorial: Your First RA Multicore Application - Blinky . . . 113
3.4.1 Introduction . . . 113
3.4.2 Generate Multicore Project Pair . . . 113
3.4.3 Information On Multicore Related Changes To hal_entry.c . . . 116
3.4.4 Build Projects . . . 117
3.4.5 Information On Multicore Related Changes To Launch Configs . . . 117
3.4.6 Reset TrustZone Partition Boundaries . . . 118
3.4.7 Start Debugging . . . 119
3.4.8 Multicore Debugging Control . . . 121
3.4.9 Terminating a Multicore Debugging Session . . . 122
3.4.10 Starting Future Debug Sessions . . . 122
3.5 Tutorial: Using HAL Drivers - Programming the WDT . . . 122
3.5.1 Application WDT . . . 123
3.5.2 Creating a WDT Application Using the RA MCU FSP and e² studio . . . 123
3.5.2.1 Using FSP and e² studio . . . 123
3.5.2.2 The WDT Application . . . 123
3.5.2.3 WDT Application flow . . . 123
3.5.3 Creating the Project with e² studio . . . 124
3.5.4 Configuring the Project with e² studio . . . 127
3.5.4.1 BSP Tab . . . 128
3.5.4.2 Clocks Tab . . . 128
3.5.4.3 Interrupts Tab . . . 129
3.5.4.4 Event Links Tab . . . 129
3.5.4.5 Pins Tab . . . 129
3.5.4.6 Stacks Tab . . . 129
3.5.4.7 Components Tab . . . 132
3.5.5 WDT Generated Project Files . . . 133
3.5.5.1 WDT hal_data.h . . . 135
3.5.5.2 WDT hal_data.c . . . 136
3.5.5.3 WDT main.c . . . 137
3.5.5.4 WDT hal_entry.c . . . 137
3.5.6 Building and Testing the Project . . . 140
3.6 Primer: Arm TrustZone Project Development . . . 141
3.6.1 Target Device . . . 142
3.6.2 Renesas Implementation of Arm TrustZone Technology . . . 142
3.6.2.1 Calling from Non-Secure to Secure . . . 143
3.6.2.2 Calling from Secure to Non-Secure . . . 143
3.6.3 Workflow . . . 144
3.6.3.1 Secure Project . . . 144
3.6.3.2 Non-Secure Project . . . 144
3.6.3.3 Flat Project . . . 144
3.6.4 RA Project Generator (PG) . . . 145
3.6.4.1 Secure Project Set Up . . . 146
3.6.4.2 RTOS Support in TrustZone Project . . . 147
3.6.4.3 Peripheral Security Attribution . . . 148
3.6.4.4 Non-Secure . . . 149
3.6.4.5 Flat Project Type . . . 149

3.6.4.6 Secure Connection to Non-Secure Project . . . 149
3.6.4.7 Debug Configurations . . . 150
3.6.5 Secure Projects . . . 151
3.6.5.1 Secure Clock . . . 151
3.6.5.2 Setting Drivers as NSC . . . 151
3.6.5.3 Guard Functions . . . 152
3.6.6 Non-Secure projects . . . 153
3.6.6.1 Clock Set Up . . . 153
3.6.6.2 Selecting NSC Drivers . . . 153
3.6.6.3 Locked Resources . . . 154
3.6.6.4 Locked Channels . . . 154
3.6.7 IDAU registers . . . 154
3.6.7.1 SCI Boot Mode . . . 156
3.6.7.2 DLM States . . . 157
3.6.7.3 Devices with Alternate DLM States . . . 158
3.6.7.4 Devices without DLM . . . 160
3.6.8 Debug . . . 160
3.6.8.1 Non-Secure Debug . . . 160
3.6.9 Debugger support . . . 161
3.6.10 Third-Party IDEs . . . 162
3.6.11 Renesas Flash Programmer (RFP) . . . 162
3.6.12 Glossary . . . 163
3.6.12.1 Configurator Icon Glossary . . . 164
3.7 RASC User Guide for MDK and IAR . . . 164
3.7.1 What is RASC? . . . 164
3.7.2 Using RA Smart Configurator with Keil MDK . . . 164
3.7.2.1 Prerequisites . . . 164
3.7.2.2 Create new RA project . . . 165
3.7.2.3 Modify existing RA project . . . 166
3.7.2.4 Build and Debug RA project . . . 167
3.7.2.5 Notes and Restrictions . . . 168
3.7.3 Using RA Smart Configurator with IAR EWARM . . . 168
3.7.3.1 Prerequisites . . . 168
3.7.3.2 Create new RA project . . . 168
3.7.3.3 Notes and Restrictions . . . 170
**Chapter 4 FSP Architecture** . . . 171
4.1 FSP Architecture Overview . . . 171
4.1.1 C99 Use . . . 171
4.1.2 Doxygen . . . 171
4.1.3 Weak Symbols . . . 171
4.1.4 Memory Allocation . . . 171
4.1.5 FSP Terms . . . 171
4.2 FSP Modules . . . 173
4.2.1 Module Sources . . . 174
4.2.2 Module Distribution . . . 174
4.2.3 Module Versioning . . . 174
4.3 FSP Stacks . . . 174
4.4 FSP Interfaces . . . 175
4.4.1 FSP Interface Enumerations . . . 176
4.4.2 FSP Interface Callback Functions . . . 176
4.4.3 FSP Interface Data Structures . . . 178
4.4.3.1 FSP Interface Configuration Structure . . . 179
4.4.3.2 FSP Interface API Structure . . . 179
4.4.3.3 FSP Interface Instance Structure . . . 181

4.5 FSP Instances . . . 182
4.5.1 FSP Instance Control Structure . . . 182
4.5.2 FSP Interface Extensions . . . 183
4.5.2.1 FSP Extended Configuration Structure . . . 183
4.5.3 FSP Instance API . . . 183
4.6 FSP API Standards . . . 183
4.6.1 FSP Function Names . . . 183
4.6.2 Use of const in API parameters . . . 184
4.6.3 FSP Version Information . . . 184
4.7 FSP Build Time Configurations . . . 184
4.8 FSP File Structure . . . 185
4.9 FSP TrustZone Support . . . 185
4.9.1 FSP TrustZone Projects . . . 185
4.9.2 Non-Secure Callable Guard Functions . . . 185
4.9.3 Callbacks in Non-Secure from Non-Secure Callable Modules . . . 186
4.9.4 Migrating TrustZone Project to newer FSP Version . . . 186
4.9.5 Additional TrustZone Information . . . 186
4.10 FSP Architecture in Practice . . . 186
4.10.1 FSP Connecting Layers . . . 186
4.10.2 Using FSP Modules in an Application . . . 187
4.10.2.1 Create a Module Instance in the RA Configuration Editor . . . 187
4.10.2.2 Use the Instance API in the Application . . . 187
**Chapter 5 API Reference** . . . 189
5.1 BSP . . . 190
5.1.1 BSP I/O access . . . 192
5.1.2 BSP OSPI_B support . . . 207
5.1.3 BSP SDRAM support . . . 207
5.1.4 Common Error Codes . . . 208
5.1.5 MCU Board Support Package . . . 220
5.1.5.1 RA0E1 . . . 269
5.1.5.2 RA0E2 . . . 271
5.1.5.3 RA0L1 . . . 273
5.1.5.4 RA2A1 . . . 275
5.1.5.5 RA2A2 . . . 276
5.1.5.6 RA2E1 . . . 278
5.1.5.7 RA2E2 . . . 280
5.1.5.8 RA2E3 . . . 281
5.1.5.9 RA2L1 . . . 283
5.1.5.10 RA2L2 . . . 286
5.1.5.11 RA2T1 . . . 288
5.1.5.12 RA4C1 . . . 290
5.1.5.13 RA4E1 . . . 294
5.1.5.14 RA4E2 . . . 299
5.1.5.15 RA4L1 . . . 304
5.1.5.16 RA4M1 . . . 308
5.1.5.17 RA4M2 . . . 310
5.1.5.18 RA4M3 . . . 315
5.1.5.19 RA4T1 . . . 321
5.1.5.20 RA4W1 . . . 326
5.1.5.21 RA6E1 . . . 327
5.1.5.22 RA6E2 . . . 333
5.1.5.23 RA6M1 . . . 338
5.1.5.24 RA6M2 . . . 340
5.1.5.25 RA6M3 . . . 344
5.1.5.26 RA6M4 . . . 347

5.1.5.27 RA6M5 . . . 353
5.1.5.28 RA6T1 . . . 358
5.1.5.29 RA6T2 . . . 360
5.1.5.30 RA6T3 . . . 365
5.1.5.31 RA8D1 . . . 370
5.1.5.32 RA8D2 . . . 380
5.1.5.33 RA8E1 . . . 381
5.1.5.34 RA8E2 . . . 389
5.1.5.35 RA8M1 . . . 400
5.1.5.36 RA8M2 . . . 410
5.1.5.37 RA8P1 . . . 411
5.1.5.38 RA8T1 . . . 412
5.1.5.39 RA8T2 . . . 420
5.2 Modules . . . 421
5.2.1 Analog . . . 424
5.2.1.1 ADC (r_adc) . . . 425
5.2.1.2 ADC (r_adc_b) . . . 462
5.2.1.3 ADC (r_adc_d) . . . 498
5.2.1.4 Comparator, High-Speed (r_acmphs) . . . 521
5.2.1.5 Comparator, High-Speed (r_acmphs_b) . . . 528
5.2.1.6 Comparator, Low-Power (r_acmplp) . . . 534
5.2.1.7 DAC (r_dac) . . . 542
5.2.1.8 DAC8 (r_dac8) . . . 549
5.2.1.9 DAC_B (r_dac_b) . . . 555
5.2.1.10 Delta-Sigma Modulator Interface (r_dsmif) . . . 560
5.2.1.11 Operational Amplifier (r_opamp) . . . 593
5.2.1.12 SDADC Channel Configuration (r_sdadc) . . . 611
5.2.1.13 SDADC_B Channel Configuration (r_sdadc_b) . . . 633
5.2.2 AI . . . 650
5.2.2.1 Ethos-U (rm_ethosu) . . . 651
5.2.2.2 Reality AI Data Collector (rm_rai_data_collector) . . . 659
5.2.2.3 Reality AI Data Shipper (rm_rai_data_shipper) . . . 673
5.2.3 Audio . . . 678
5.2.3.1 ADPCM Decoder (rm_adpcm_decoder) . . . 679
5.2.3.2 Audio Playback PWM (rm_audio_playback_pwm) . . . 683
5.2.3.3 Minimp3 Library . . . 695
5.2.3.4 Pulse Density Modulation Interface (r_pdm) . . . 698
5.2.4 Bootloader . . . 720
5.2.4.1 MCUboot Port (rm_mcuboot_port) . . . 720
5.2.5 CapTouch . . . 742
5.2.5.1 CTSU (r_ctsu) . . . 743
5.2.5.2 Touch (rm_touch) . . . 787
5.2.6 Connectivity . . . 809
5.2.6.1 Azure RTOS USBX Port (rm_usbx_port) . . . 815
5.2.6.2 CAN (r_can) . . . 931
5.2.6.3 CAN FD (r_canfd) . . . 969
5.2.6.4 CEC (r_cec) . . . 1005
5.2.6.5 I2C Communication Device (rm_comms_i2c) . . . 1016
5.2.6.6 I2C Master (r_iic_b_master) . . . 1024
5.2.6.7 I2C Master (r_iic_master) . . . 1038
5.2.6.8 I2C Master (r_iica_master) . . . 1052
5.2.6.9 I2C Master (r_sau_i2c) . . . 1064
5.2.6.10 I2C Master (r_sci_b_i2c) . . . 1080
5.2.6.11 I2C Master (r_sci_i2c) . . . 1093
5.2.6.12 I2C Slave (r_iic_b_slave) . . . 1106

5.2.6.13 I2C Slave (r_iic_slave) . . . . . 1118
5.2.6.14 I2C Slave (r_iica_slave) . . . . . 1129
5.2.6.15 I2S (r_ssi) . . . . . 1140
5.2.6.16 I3C (r_i3c) . . . . . 1155
5.2.6.17 LIN (r_sau_lin) . . . . . 1197
5.2.6.18 LIN (r_sci_b_lin) . . . . . 1221
5.2.6.19 LIN (r_sci_lin) . . . . . 1257
5.2.6.20 SMBUS Communication Device (rm_comms_smbus) . . . . . 1291
5.2.6.21 SMCI (r_sci_smci) . . . . . 1302
5.2.6.22 SPI (r_sau_spi) . . . . . 1316
5.2.6.23 SPI (r_sci_b_spi) . . . . . 1330
5.2.6.24 SPI (r_sci_spi) . . . . . 1342
5.2.6.25 SPI (r_spi) . . . . . 1354
5.2.6.26 SPI (r_spi_b) . . . . . 1372
5.2.6.27 SPI Communication Device (rm_comms_spi) . . . . . 1390
5.2.6.28 UART (r_sau_uart) . . . . . 1398
5.2.6.29 UART (r_sci_b_uart) . . . . . 1415
5.2.6.30 UART (r_sci_uart) . . . . . 1434
5.2.6.31 UART (r_uarta) . . . . . 1456
5.2.6.32 UART Communication Device (rm_comms_uart) . . . . . 1472
5.2.6.33 USB (r_usb_basic) . . . . . 1479
5.2.6.34 USB Composite (r_usb_composite) . . . . . 1541
5.2.6.35 USB HAUD (r_usb_haud) . . . . . 1571
5.2.6.36 USB HCDC (r_usb_hcdc) . . . . . 1592
5.2.6.37 USB HCDC_ECM (r_usb_hcdc_ecm) . . . . . 1604
5.2.6.38 USB HHID (r_usb_hhid) . . . . . 1625
5.2.6.39 USB HMSC (r_usb_hmsc) . . . . . 1635
5.2.6.40 USB Host Vendor class (r_usb_hvnd) . . . . . 1645
5.2.6.41 USB PAUD (r_usb_paud) . . . . . 1658
5.2.6.42 USB PCDC (r_usb_pcdc) . . . . . 1701
5.2.6.43 USB PHID (r_usb_phid) . . . . . 1710
5.2.6.44 USB PMSC (r_usb_pmsc) . . . . . 1722
5.2.6.45 USB PPRN (r_usb_pprn) . . . . . 1728
5.2.6.46 USB Peripheral Vendor class (r_usb_pvnd) . . . . . 1736
5.2.6.47 USB-TYPEC (r_usb_typec) . . . . . 1747
5.2.6.48 USB_PCDC Communication Device (rm_comms_usb_pcdc) . . . . . 1754
5.2.7 DSP . . . . . 1762
5.2.7.1 CMSIS DSP H/W Acceleration (rm_cmsis_dsp) . . . . . 1763
5.2.7.2 IIR Filter Accelerator (r_iirfa) . . . . . 1764
5.2.8 Graphics . . . . . 1772
5.2.8.1 Azure RTOS GUIX Port (rm_guix_port) . . . . . 1774
5.2.8.2 Capture Engine Unit (r_ceu) . . . . . 1781
5.2.8.3 D/AVE 2D Port Interface (r_drw) . . . . . 1795
5.2.8.4 Display Serial Interface (r_vin) . . . . . 1797
5.2.8.5 Graphics LCD (r_glcdc) . . . . . 1810
5.2.8.6 JPEG Codec (r_jpeg) . . . . . 1849
5.2.8.7 LVGL Port (rm_lvgl_port) . . . . . 1876
5.2.8.8 MIPI Display Serial Interface (r_mipi_dsi) . . . . . 1888
5.2.8.9 Parallel Data Capture (r_pdc) . . . . . 1908
5.2.8.10 SEGGER emWin RA Port (rm_emwin_port) . . . . . 1919
5.2.8.11 Segment LCD (r_slcdc) . . . . . 1926
5.2.9 Input . . . . . 1935
5.2.9.1 External IRQ (r_icu) . . . . . 1936
5.2.9.2 Key Matrix (r_kint) . . . . . 1943

5.2.10 Monitoring . . . 1947
5.2.10.1 CRC (r_crc) . . . 1948
5.2.10.2 Clock Accuracy Circuit (r_cac) . . . 1954
5.2.10.3 Data Operation Circuit (r_doc) . . . 1960
5.2.10.4 Independent Watchdog (r_iwdt) . . . 1967
5.2.10.5 Low/Programmable Voltage Detection (r_lvd) . . . 1977
5.2.10.6 Watchdog (r_wdt) . . . 1985
5.2.11 Motor . . . 1997
5.2.11.1 120-degree conduction control sensorless (rm_motor_120_control_sensorless) . . . 1999
5.2.11.2 120-degree conduction control with Hall sensors (rm_motor_120_control_hall) . . . 2017
5.2.11.3 ADC and PWM Modulation (rm_motor_driver) . . . 2032
5.2.11.4 ADC and PWM modulation (rm_motor_120_driver) . . . 2042
5.2.11.5 Motor 120 degree control (rm_motor_120_degree) . . . 2057
5.2.11.6 Motor Angle (rm_motor_estimate) . . . 2078
5.2.11.7 Motor Angle (rm_motor_sense_encoder) . . . 2088
5.2.11.8 Motor Angle and Speed Calculation with Hall sensors (rm_motor_sense_hall) . . . 2100
5.2.11.9 Motor Angle and Speed Calculation with induction sensor (rm_motor_sense_induction) . . . 2110
5.2.11.10 Motor Current Controller (rm_motor_current) . . . 2122
5.2.11.11 Motor Encoder Vector Control (rm_motor_encoder) . . . 2133
5.2.11.12 Motor Inertia estimate (rm_motor_inertia_estimate) . . . 2155
5.2.11.13 Motor Position Controller (rm_motor_position) . . . 2165
5.2.11.14 Motor Sensorless Vector Control (rm_motor_sensorless) . . . 2176
5.2.11.15 Motor Speed Controller (rm_motor_speed) . . . 2194
5.2.11.16 Motor Vector Control with hall sensors (rm_motor_hall) . . . 2206
5.2.11.17 Motor return origin (rm_motor_return_origin) . . . 2223
5.2.11.18 Motor vector control with induction sensor (rm_motor_induction) . . . 2232
5.2.12 Networking . . . 2253
5.2.12.1 AWS Cellular Interface on GM (rm_cellular_gm_aws) . . . 2257
5.2.12.2 AWS MQTT . . . 2264
5.2.12.3 AWS OTA PAL on MCUBoot (rm_aws_ota_pal_mcuboot) . . . 2271
5.2.12.4 AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs) . . . 2274
5.2.12.5 AWS coreHTTP . . . 2274
5.2.12.6 Azure Embedded Wireless Framework GM Port (rm_azure_ewf_gm) . . . 2280
5.2.12.7 BLE Abstraction (rm_ble_abs) . . . 2286
5.2.12.8 BLE Driver (r_ble_balance) . . . 2323
5.2.12.9 BLE Driver (r_ble_compact) . . . 2326
5.2.12.10 BLE Driver (r_ble_extended) . . . 2329
5.2.12.11 BLE Mesh Network Modules . . . 2332
5.2.12.12 Cellular Comm Interface on UART (rm_cellular_comm_uart_aws) . . . 2894
5.2.12.13 DA16XXX Transport Layer (rm_at_transport_da16xxx_uart) . . . 2899
5.2.12.14 EtherCAT SSC Port (rm_ethercat_ssc_port) . . . 2907
5.2.12.15 Ethercat (r_ethercat_phy) . . . 2913
5.2.12.16 Ethernet (r_ether) . . . 2923
5.2.12.17 Ethernet (r_ether_phy) . . . 2948
5.2.12.18 Ethernet (r_rmac) . . . 2959
5.2.12.19 Ethernet (r_rmac_phy) . . . 2974
5.2.12.20 Ethernet Generic PTP Timer (r_gptp) . . . 2988
5.2.12.21 FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp) . . . 2998
5.2.12.22 GTL BLE Abstraction (rm_ble_abs_gtl) . . . 3005
5.2.12.23 Layer3 Ethernet Switch (r_layer3_switch) . . . 3016
5.2.12.24 NFC Reader Driver on PTX (rm_nfc_reader_ptx) . . . 3064
5.2.12.25 NetX Duo Ethernet Driver (rm_netxduo_ether) . . . 3084
5.2.12.26 NetX Duo WiFi Driver (rm_netxduo_wifi) . . . 3088
5.2.12.27 On Chip HTTP Client on DA16XXX (rm_http_onchip_da16xxx) . . . 3095
5.2.12.28 On Chip MQTT Client on DA16XXX (rm_mqtt_onchip_da16xxx) . . . 3106

5.2.12.29 PTP (r_ptp) . . . 3125
5.2.12.30 SPP BLE Abstraction (rm_ble_abs_spp) . . . 3147
5.2.12.31 Socket Communication Device (rm_comms_socket) . . . 3159
5.2.12.32 WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx) . . . 3168
5.2.12.33 WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex) . . . 3180
5.2.12.34 lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal) . . . 3205
5.2.12.35 lwIP Ethernet Driver (rm_lwip_ether) . . . 3206
5.2.12.36 lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos) . . . 3210
5.2.12.37 lwIP HTTP Server . . . 3211
5.2.12.38 lwIP MQTT . . . 3220
5.2.13 Power . . . 3227
5.2.13.1 Low Power Modes (r_lpm) . . . 3227
5.2.14 RTOS . . . 3237
5.2.14.1 Azure RTOS ThreadX Port (rm_threadx_port) . . . 3237
5.2.14.2 FreeRTOS Port (rm_freertos_port) . . . 3247
5.2.15 Security . . . 3276
5.2.15.1 AWS Device Provisioning . . . 3277
5.2.15.2 Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto) . . . 3281
5.2.15.3 Mbed Crypto H/W Acceleration (rm_psa_crypto) . . . 3321
5.2.15.4 Ocrypto (rm_ocrypto_port) . . . 3399
5.2.15.5 Renesas Secure IP (r_rsip_e11a_protected_cavp) . . . 3404
5.2.15.6 Renesas Secure IP (r_rsip_e51a_protected_cavp) . . . 3442
5.2.15.7 Renesas Secure IP (r_rsip_protected) . . . 3490
5.2.15.8 SCE Protected Mode . . . 3677
5.2.15.9 Secure Crypto Engine (r_sce_protected_cavp) . . . 3839
5.2.15.10 Secure Key Injection (r_rsip_key_injection) . . . 3848
5.2.15.11 Secure Key Injection (r_sce_key_injection) . . . 3878
5.2.15.12 TinyCrypt H/W Acceleration (rm_tinycrypt_port) . . . 3917
5.2.16 Sensor . . . 3941
5.2.16.1 FS3000 Flow Sensor (rm_fs3000) [Deprecated] . . . 3942
5.2.16.2 HS300X Temperature/Humidity Sensor (rm_hs300x) [Deprecated] . . . 3949
5.2.16.3 HS400X Temperature/Humidity Sensor (rm_hs400x) [Deprecated] . . . 3961
5.2.16.4 RRH46410 Gas Sensor Module (rm_rrh46410) [Deprecated] . . . 3974
5.2.16.5 RRH47000 NDIR CO2 Sensor Module (rm_rrh47000) [Deprecated] . . . 3995
5.2.16.6 RRH62000 All-in-one Air Quality Module (rm_rrh62000) [Deprecated] . . . 4005
5.2.16.7 ZMOD4XXX Gas Sensor (rm_zmod4xxx) [Deprecated] . . . 4016
5.2.17 Storage . . . 4101
5.2.17.1 Block Media Custom Implementation (rm_block_media_user) . . . 4103
5.2.17.2 Block Media RAM (rm_block_media_ram) . . . 4105
5.2.17.3 Block Media SD/MMC (rm_block_media_sdmmc) . . . 4112
5.2.17.4 Block Media SPI Flash (rm_block_media_spi) . . . 4119
5.2.17.5 Block Media USB (rm_block_media_usb) . . . 4128
5.2.17.6 FileX I/O (rm_filex_block_media) . . . 4135
5.2.17.7 FileX I/O (rm_filex_levelx_nor) . . . 4153
5.2.17.8 Flash (r_flash_hp) . . . 4168
5.2.17.9 Flash (r_flash_lp) . . . 4196
5.2.17.10 FreeRTOS+FAT Port for RA (rm_freertos_plus_fat) . . . 4216
5.2.17.11 LevelX NOR Port (rm_levelx_nor_spi) . . . 4229
5.2.17.12 LittleFS on Flash (rm_littlefs_flash) . . . 4243
5.2.17.13 LittleFS on SPI Flash (rm_littlefs_spi_flash) . . . 4250
5.2.17.14 MRAM (r_mram) . . . 4257
5.2.17.15 OSPI Flash (r_ospi) . . . 4272
5.2.17.16 OSPI Flash (r_ospi_b) . . . 4299
5.2.17.17 QSPI (r_qspi) . . . 4345
5.2.17.18 SD/MMC (r_sdhi) . . . 4363

5.2.17.19 Virtual EEPROM on Flash (rm_vee_flash) . . . . 4377
5.2.18 System . . . . 4390
5.2.18.1 Clock Generation Circuit (r_cgc) . . . . 4391
5.2.18.2 Event Link Controller (r_elc) . . . . 4415
5.2.18.3 I/O Port (r_ioport) . . . . 4423
5.2.18.4 Inter-Processor Communication (r_ipc) . . . . 4460
5.2.19 Timers . . . . 4467
5.2.19.1 Independent Channel, 16-bit and 8-bit timer (r_tau) . . . . 4468
5.2.19.2 Port Output Enable for GPT (r_poeg) . . . . 4495
5.2.19.3 Realtime Clock (r_rtc) . . . . 4503
5.2.19.4 Realtime Clock (r_rtc_c) . . . . 4521
5.2.19.5 Three-Phase PWM (r_gpt_three_phase) . . . . 4534
5.2.19.6 Timer, 32-bit Interval Timer (r_tml) . . . . 4543
5.2.19.7 Timer, General PWM (r_gpt) . . . . 4562
5.2.19.8 Timer, Low-Power (r_agt) . . . . 4614
5.2.19.9 Timer, Simultaneous Channel (r_tau_pwm) . . . . 4641
5.2.19.10 Timer, Ultra Low-Power (r_ulpt) . . . . 4664
5.2.20 Transfer . . . . 4687
5.2.20.1 Transfer (r_dmac) . . . . 4688
5.2.20.2 Transfer (r_dtc) . . . . 4702
5.2.21 TrustZone . . . . 4715
5.2.21.1 Arm TrustZone Context RA Port (rm_tz_context) . . . . 4715
5.3 Interfaces . . . . 4716
5.3.1 Analog . . . . 4718
5.3.1.1 ADC Interface . . . . 4719
5.3.1.2 Comparator Interface . . . . 4747
5.3.1.3 DAC Interface . . . . 4755
5.3.1.4 OPAMP Interface . . . . 4759
5.3.2 AI . . . . 4764
5.3.2.1 Data Collector Interface . . . . 4765
5.3.2.2 Data Shipper Interface . . . . 4772
5.3.2.3 Ethos-U Interface . . . . 4776
5.3.3 Audio . . . . 4780
5.3.3.1 ADPCM Decoder Interface . . . . 4780
5.3.3.2 AUDIO PLAYBACK Interface . . . . 4783
5.3.3.3 PDM Interface . . . . 4787
5.3.4 CapTouch . . . . 4797
5.3.4.1 CTSU Interface . . . . 4798
5.3.4.2 Touch Middleware Interface . . . . 4813
5.3.5 Connectivity . . . . 4821
5.3.5.1 CAN Interface . . . . 4823
5.3.5.2 CEC Interface . . . . 4833
5.3.5.3 Communications Middleware Interface . . . . 4841
5.3.5.4 I2C Master Interface . . . . 4846
5.3.5.5 I2C Slave Interface . . . . 4854
5.3.5.6 I2S Interface . . . . 4862
5.3.5.7 I3C Interface . . . . 4873
5.3.5.8 LIN Interface . . . . 4890
5.3.5.9 SMCI Interface . . . . 4899
5.3.5.10 SPI Interface . . . . 4911
5.3.5.11 UART Interface . . . . 4922
5.3.5.12 USB HAUD Interface . . . . 4934
5.3.5.13 USB HCDC Interface . . . . 4937
5.3.5.14 USB HHID Interface . . . . 4943
5.3.5.15 USB HMSC Interface . . . . 4945

5.3.5.16 USB Interface ..... 4950
5.3.5.17 USB PAUD Interface ..... 4977
5.3.5.18 USB PCDC Interface ..... 4979
5.3.5.19 USB PHID Interface ..... 4982
5.3.5.20 USB PMSC Interface ..... 4982
5.3.5.21 USB PPRN Interface ..... 4982
5.3.5.22 USB TYPEC Interface ..... 4983
5.3.6 DSP ..... 4989
5.3.6.1 IIR Interface ..... 4989
5.3.7 Graphics ..... 4994
5.3.7.1 CAPTURE Interface ..... 4994
5.3.7.2 Display Interface ..... 5000
5.3.7.3 JPEG Codec Interface ..... 5017
5.3.7.4 MIPI CMD Data Types ..... 5030
5.3.7.5 MIPI DSI Interface ..... 5032
5.3.7.6 SLCDC Interface ..... 5059
5.3.8 Input ..... 5070
5.3.8.1 External IRQ Interface ..... 5071
5.3.8.2 Key Matrix Interface ..... 5076
5.3.9 Monitoring ..... 5080
5.3.9.1 CAC Interface ..... 5081
5.3.9.2 CRC Interface ..... 5091
5.3.9.3 DOC Interface ..... 5096
5.3.9.4 Low Voltage Detection Interface ..... 5102
5.3.9.5 WDT Interface ..... 5123
5.3.10 Motor ..... 5132
5.3.10.1 Motor 120-Degree Control Interface ..... 5133
5.3.10.2 Motor 120-Degree Driver Interface ..... 5144
5.3.10.3 Motor Inertia Estimate Interface ..... 5152
5.3.10.4 Motor Interface ..... 5157
5.3.10.5 Motor Return Origin Function Interface ..... 5165
5.3.10.6 Motor angle Interface ..... 5170
5.3.10.7 Motor current Interface ..... 5178
5.3.10.8 Motor driver Interface ..... 5186
5.3.10.9 Motor position Interface ..... 5192
5.3.10.10 Motor speed Interface ..... 5199
5.3.11 Networking ..... 5205
5.3.11.1 BLE ABS Interface ..... 5206
5.3.11.2 BLE Interface ..... 5240
5.3.11.3 BLE Mesh Network Interfaces ..... 5611
5.3.11.4 DA16XXX AT Command Transport Layer ..... 5807
5.3.11.5 EtherCAT SSC Interface ..... 5813
5.3.11.6 Ethernet Interface ..... 5817
5.3.11.7 Ethernet PHY Interface ..... 5827
5.3.11.8 Ethernet Switch Interface ..... 5834
5.3.11.9 GPTP Interface ..... 5838
5.3.11.10 PTP Interface ..... 5843
5.3.11.11 WiFi Interface ..... 5868
5.3.12 Power ..... 5904
5.3.12.1 Low Power Modes Interface ..... 5904
5.3.13 Security ..... 5944
5.3.13.1 RSIP Interface ..... 5945
5.3.13.2 RSIP key injection Interface ..... 5999
5.3.13.3 SCE Interface ..... 6027
5.3.13.4 SCE key injection Interface ..... 6117

5.3.14 Sensor . . . . . 6138
5.3.14.1 FSXXXX Middleware Interface . . . . . 6139
5.3.14.2 HS300X Middleware Interface . . . . . 6143
5.3.14.3 HS400X Middleware Interface . . . . . 6149
5.3.14.4 SENSOR Middleware Interface . . . . . 6155
5.3.14.5 ZMOD4XXX Middleware Interface . . . . . 6162
5.3.15 Storage . . . . . 6174
5.3.15.1 Block Media Interface . . . . . 6175
5.3.15.2 FileX Block Media Port Interface . . . . . 6183
5.3.15.3 Flash Interface . . . . . 6186
5.3.15.4 FreeRTOS+FAT Port Interface . . . . . 6203
5.3.15.5 LittleFS Interface . . . . . 6209
5.3.15.6 SD/MMC Interface . . . . . 6211
5.3.15.7 SPI Flash Interface . . . . . 6228
5.3.15.8 Virtual EEPROM Interface . . . . . 6241
5.3.16 System . . . . . 6248
5.3.16.1 CGC Interface . . . . . 6249
5.3.16.2 ELC Interface . . . . . 6265
5.3.16.3 I/O Port Interface . . . . . 6269
5.3.16.4 IPC Interface . . . . . 6276
5.3.17 Timers . . . . . 6282
5.3.17.1 POEG Interface . . . . . 6283
5.3.17.2 RTC Interface . . . . . 6292
5.3.17.3 Three-Phase Interface . . . . . 6308
5.3.17.4 Timer Interface . . . . . 6313
5.3.18 Transfer . . . . . 6329
5.3.18.1 Transfer Interface . . . . . 6329
**Chapter 6 Copyright** . . . . . 6342

# Chapter 1 Introduction

## 1.1 Overview

This manual describes how to use the Renesas Flexible Software Package (FSP) for writing applications for the RA microcontroller series.

## 1.2 Using this Manual

This manual provides a wide variety of information, so it can be helpful to know where to start. Here is a short description of each main section and how they can be used.

Starting Development - Provides a step by step guide on how to use e² studio and FSP to develop a project for RA MCUs. This is a good place to start to get up to speed quickly and efficiently.

FSP Architecture - Provides useful background material on key FSP concepts such as Modules, Stacks, and API standards. Reference this section to extend or refresh your knowledge of FSP concepts.

**API Reference** - Provides detailed information on each module and interface including features, API functions, configuration settings, usage notes, function prototypes and code examples. Board Support Package (BSP) related API functions are also included.

*Note*

*Much of the information in the API Reference section is available from within the e² studio tool via the Developer Assistance feature. The information here can be referenced for additional details on API features.*

## 1.3 Documentation Standard

Each Modules section user guide outlines the following:

- Features: A bullet list of high level features provided by the module.
- Configuration: A description of module specific configurations available in the RA Configuration editor.
- Usage Notes: Module specific documentation and limitations.
- Examples: Example code provided to help the user get started.
- API Reference: Usage notes for each API in the module, including the function prototype and hyperlinks to the interface documentation for parameter definitions.

Each Interfaces section user guide outlines the following:

- Detailed Description: A short description and summary of the interface functionality.
- Data Structures: A list and definition of each data structure used by the interface including the structure of the pointers that define the API and are shared by all modules that implement the interface.
- Typedefs: A list and description of the typedefs used by the interface.
- Enumerations: A list and description of the enumerations used by the interface.

# 1.4 Introduction to FSP

## 1.4.1 Purpose

The Renesas Flexible Software Package (FSP) is an optimized software package designed to provide easy to use, scalable, high quality software for embedded system design. The primary goal is to provide lightweight, efficient drivers that meet common use cases in embedded systems.

## 1.4.2 Quality

FSP code quality is enforced by peer reviews, automated requirements-based testing, and automated static analysis.

## 1.4.3 Ease of Use

FSP provides uniform and intuitive APIs that are well documented. Each module is supported with detailed user documentation including example code.

## 1.4.4 Scalability

FSP modules can be used on any MCU in the RA family, provided the MCU has any peripherals required by the module.

## 1.4.5 Build Time Configurations

FSP modules also have build time configurations that can be used to optimize the size of the module for the feature set required by the application.

## 1.4.6 e² studio IDE

FSP provides a host of efficiency enhancing tools for developing projects targeting the Renesas RA series of MCU devices. The e² studio IDE provides a familiar development cockpit from which the key steps of project creation, module selection and configuration, code development, code generation, and debugging are all managed.

# Chapter 2 Reference Materials

This section contains miscellaneous reference materials.

| Document | Summary |
|---|---|
| Using Registers Directly | Overview of register definition files and their use. |
| FSP v4.0.0 FreeRTOS Stack Migration Guide | Migration guide for FreeRTOS Network stack users moving to FSP v4.0.0. |
| Cortex-M85 Caches | Overview of the Cortex-M85 caches present in RA8 devices and when to perform cache maintenance. |
| Using Ocrypto | Information for the use of Ocrypto |
| Linker Script Generation Guide | Linker section naming guide |

## 2.1 Using Registers Directly

### 2.1.1 Overview

In some cases users may wish to utilize chip functionality that is not supported yet by FSP. While we encourage contacting us when new feature requests arise it takes time before any updates are made. In the meantime, it is recommended to use the register definition files accessed via renesas.h to add custom functionality as needed. Official ARM CMSIS documentation refers to this format of related register definition files as device header files. Within Renesas, these files are often referred to as an **IO define** or more commonly **iodefine**.

### 2.1.2 What's in an "iodefine"?

Iodefine files contain definitions for all the I/O registers on a device. For RA FSP, one iodefine header is provided per device group (RA6M3, RA2A1 etc.) that contains all the register definitions provided in the hardware manual for that group. These headers are accessed via renesas.h, which selects the appropriate file based on the MCU configured in the project.

*Note*

*Prior to FSP 4.0.0 renesas.h contained a superset of all register definitions for RA MCUs. This became unsustainable as new device families were added and was confusing for users, so headers for each device were created and renesas.h was updated to serve as a selector.*

### 2.1.3 Where are iodefine files located?

In RA FSP, iodefine files are stored in ra/fsp/src/bsp/cmsis/Device/RENESAS/Include. This includes both renesas.h as well as the device group specific iodefine headers.

## 2.1.4 Using the register definitions

Each peripheral register set is provided as a struct. In general, the template to follow is:

```
R_ + peripheral abbreviation + channel number + -> + register name [ + _b. + bitfield
name]
```

### Basic register access

Registers are most commonly accessed whole. For example, say we want to read the counter on GPT channel 3:

```
uint32_t count = R_GPT3->GTCNT;
```

As shown in the template, the iodefines can also be used to access bitfields. For example, to start GPT channel 3:

```
R_GPT3->GTCR_b.CST = 1;
```

### Bitfield macros

It is worth noting that each bitfield access will cause a full read-modify-write which cannot be combined by the compiler because the register definitions are by necessity volatile. This can be very size and speed inefficient, particularly when the peripheral is on a very slow clock. It is recommended to write whole registers wherever possible. This is made easier by macros that are provided alongside the register definitions.

Every bitfield has two associated macros: _Msk (bitfield mask) and _Pos (bit position). These macros can be used to manipulate whole registers instead of using bitfield access when multiple bitfields are modified simultaneously. For example, setting both GPT3 output pins to 100% duty cycle:

```
uint32_t gtuddtyc = R_GPT3->GTUDDTYC
gtuddtyc         &= ~(R_GPT0_GTUDDTYC_OADTY_Msk |
R_GPT0_GTUDDTYC_OBDTY_MSK)             // Mask off bits to clear
gtuddtyc         |= (3 << R_GPT0_GTUDDTYC_OADTY_Pos) | (3 <<
R_GPT0_GTUDDTYC_OBDTY_Pos); // Shift values to bitfield positions
R_GPT3->GTUDDTYC = gtuddtyc;
```

*Note*

*The macros are all named with channel 0 due to internal process requirements; they can be used with all channels.*

### Arrays and clusters

Some registers are part of an array. These are typically listed in the manual using indexes like "n" or "m", but may occasionally be listed with individual numbers or letters. Accessing register arrays is just like accessing a regular array. For example, the GPT duty cycle registers are GTCCRA through GTCCRF. In the iodefine, they are accessed via the GTCCR array:

```
/* Set GTCCRB */
R_GPT3->GTCCR[1] = duty_cycle_b;
```

*Note*

*Not all indices may be available in every array. Be sure to check the hardware manual to verify which registers are provided on the device in use.*

Sometimes there are groups of registers that are repeated multiple times. These groups are called a **cluster**. Clusters are accessed similarly to arrays, except the actual registers are elements of the array values instead. For example, mailboxes in the CAN peripheral have several registers each:

```
/* Get data from mailbox 3 */
uint32_t mailbox = 3;
/* Get the frame data length code */
uint32_t data_length = R_CAN0->MB[mailbox].DL_b.DLC;
/* Get message data */
uint8_t data[8];
for (uint32_t i = 0U; i < data_length; i++)
{
    data[i] = p_reg->MB[mailbox].D[i]; // Copy receive data to buffer
}
```

## 2.1.5 Tips for writing hardware drivers

### Channel register access

FSP drivers often calculate a register offset in the Open function. This is typically done by multiplying the channel by the offset between channels 0 and 1. For example, here is the calculation for GPT:

```
  /* Save register base address. */
  uint32_t base_address = (uint32_t) R_GPT0 + (p_cfg->channel * ((uint32_t) R_GPT1 -
(uint32_t) R_GPT0));
  p_instance_ctrl->p_reg = (R_GPT0_Type *) base_address;
```

When drivers need to support multiple channels of a peripheral it can be helpful to save the offset in a persistent variable or structure so that it only needs to be calculated once.

# 2.2 FSP v4.0.0 FreeRTOS Stack Migration Guide

*Note*

***This guide only applies to users converting e² studio projects created with FSP v3.8.0 or earlier***
***Migration is only possible for network stacks using CoreMQTT and CoreHTTP modules. Stacks using deprecated MQTT Client and HTTP Client modules will not be able to be migrated over to FSP v4.0.0 and up.***
***See the following guide for migrating code from the deprecated Client libraries: https://aws.github.io/aws-iot-device-sdk-embedded-C/202103.00/docs/doxygen/output/html/mqtt_migration.html***

## 2.2.1 Overview

This migration guide is for moving to the new FreeRTOS Network stacks introduced in FSP v4.0.0. It will describe steps for moving over stacks as well as changes to make code compile correctly.

## 2.2.2 Stack Migration Steps

*Note*

***Perform all of the following steps before attempting to save the project. Failure to do so may cause unexpected errors in the configuration.xml.***

1. Open project to be migrated in e² studio using FSP v4.0.0 or greater. The existing CoreMQTT/HTTP stack should still exist. Many of the stack components will have No Longer Supported in the display and have permanent constraint error messages on them:

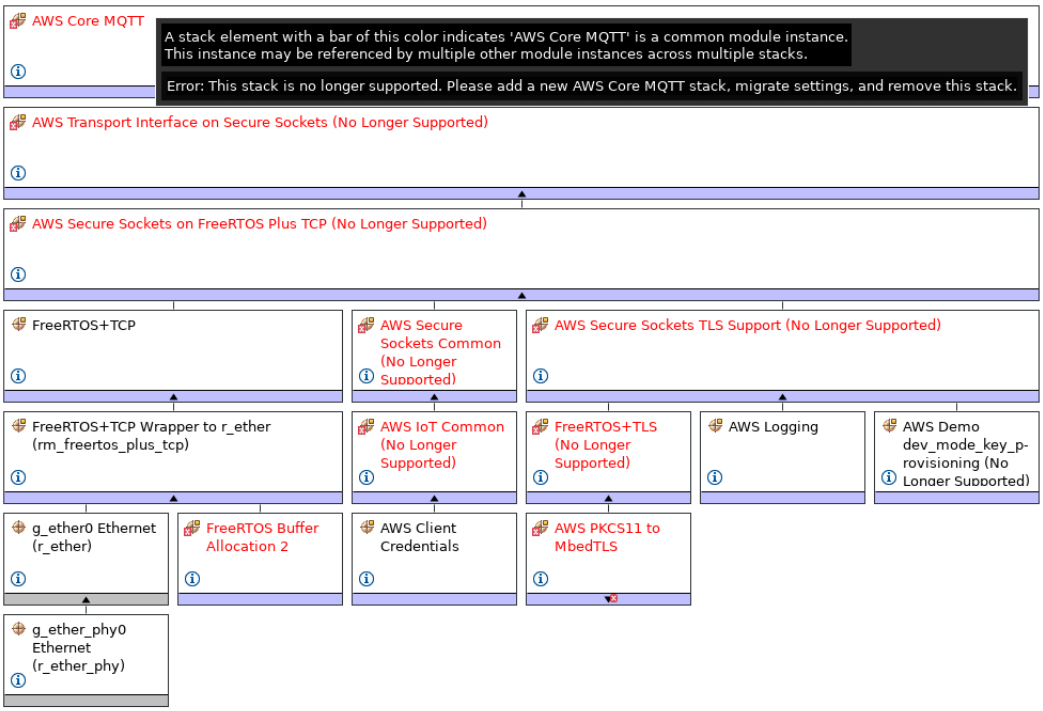



Figure 1: Old MQTT Stack

2. Add a new network stack (New Stack->Networking->AWS CoreHTTP or AWS CoreMQTT):

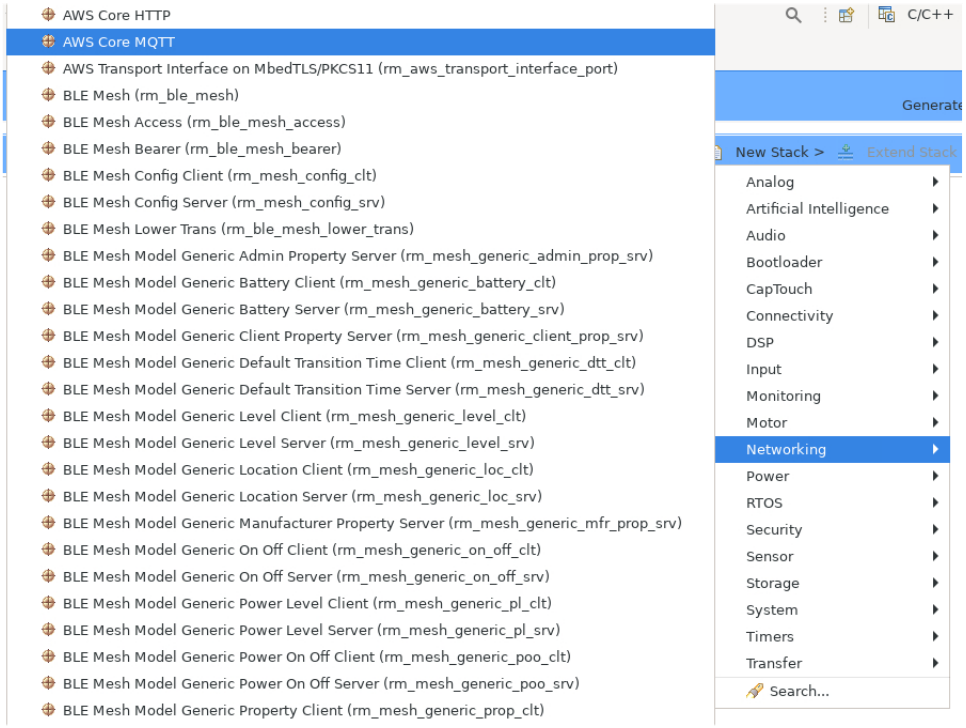



Figure 2: Add a New MQTT Stack

3. Choose a sockets wrapper for the new stack (Add Sockets Wrapper).

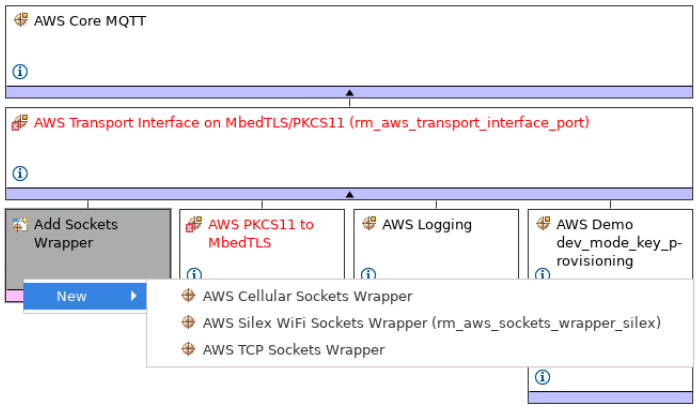



Figure 3: Add a Sockets Wrapper

4. The submodule for AWS PKCS11 to MbedTLS will likely be missing. Click Add FreeRTOS MbedTLS Port and use the existing instance:

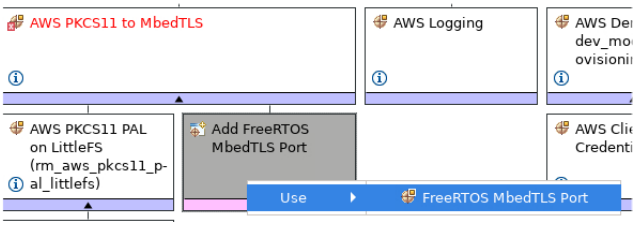



Figure 4: FreeRTOS MbedTLS Port

5. Got to the components tab and see if AWS MbedTLS FreeRTOS Port (AWS|Abstractions|FreeRTOS_Plus|utilities|mbedtls) is checked. If this component is not checked then check it so that appears like so:

Figure 5: FreeRTOS MbedTLS Port Component

6. Check that settings/properties are correct on the new stack. Common stack elements between the old and new stacks should already have previous settings.
7. Remove the old stack.
8. Save project and generate project content. See the next section for things to change in the code for successful compilation.

## 2.2.3 List of Code Change Highlights

This is a list of important changes from pre-4.0 code. For more detailed examples see CoreMQTT/HTTP examples and devassist.

- Any calls to SYSTEM_Init() should be removed. mbedtls_platform_setup() now does initialization that this IoT library function originally did.
- xLoggingTaskInitialize should no longer be called. Logging libraries which create a logging task no longer exist. Logging by default uses printf when enabled. The user can redefine vLoggingPrintf(const char * pcFormat, ...) and vLoggingPrint(const char * pcFormat) to change this behavior.
- The transport interface for CoreMQTT/HTTP will now need to be setup directly by creating a transport interface and setting the send and receive pointers to TLS_FreeRTOS_recv and TLS_FreeRTOS_send. See the examples and devassist for more info.
- The user will need to call TLS_FreeRTOS_Connect to connect to a server/endpoint before using CoreMQTT/HTTP APIs to communicate. TLS_FreeRTOS_Disconnect can be used to disconnect from a server/endpoint.

# 2.3 Cortex-M85 Caches

*Note*

***This overview should be considered supplementary information only. Consult with the listed references (non-exhaustive) for detailed information on the CM85 caches, Renesas caches, and cache coherency. Cache coherency can be a difficult problem to understand and solve correctly.***

## 2.3.1 Changes

| FSP Version | Changes |
|---|---|
| 5.6.0 | Add information about cache incoherency with memory attribute mismatch between TrustZone security states. Add information about CCR.xC TrustZone banked behavior. Add information about cache incoherency with predefined non-cacheable sections when crossing security states. Add new Arm KB reference. |
| 6.0.0 | Update information about predefined non-cacheable sections, which have changed with linker script updates. |

## 2.3.2 Overview

When using any type of caches in a system, coherency must be considered. A cache may contain data that is different from the backing memory (e.g. SRAM, Flash, etc.), or contain data that is different from another cache, which would make the cache incoherent. Coherency can be maintained through hardware and/or software support. For Cortex-M devices like the RA8, coherency can only be achieved through manual software management, and no automatic hardware coherency support exists.

In the default configuration for RA8 devices, FSP always enables the Code Flash Cache (FCACHE) and CM85 Instruction Cache (I-Cache) and handles the coherency of these caches where it is required. FSP does not handle FCACHE or I-Cache coherency outside of FSP. FSP optionally allows the CM85 Data Cache (D-Cache) to be enabled in the BSP configuration settings, where it is disabled by default. If the D-Cache is enabled, additional coherency concerns will arise. **FSP does not currently handle any D-Cache coherency. This is a work in-progress. Drivers that will require D-Cache coherency support do not presently support it.**

If D-Cache is enabled, the most common coherency concern is when data is shared between the CPU and another bus master. The D-Cache and backing memory for the shared location (usually SRAM) can become incoherent. To properly manage coherency in this situation, do one of the following:

- Place the shared data in a non-cacheable region defined by the MPU or hardware.

  Using a non-cacheable region means there will be no cache maintenance required and no coherency issues because the shared data will not be cached. A non-cacheable region can be defined in the MPU to contain the shared data. The non-cacheable region **MUST** be aligned to 32 bytes and be a length multiple of 32 bytes. This is required to meet MPU alignment and length requirements. FSP predefines several non-cacheable linker sections for this purpose, where data can be placed (see the BSP Usage Notes reference material). FSP configures and enables the MPU with these predefined regions at startup, if they have a non-zero size. Any address outside these regions will use the cacheability attributes defined by the default system address map. As an alternative, DTCM is always hardcoded as non-cacheable by the hardware and could contain the shared data. Using the MPU should be preferred, since DTCM can only be accessed through S-AHB by other bus masters and the access may contend with CPU access to DTCM.

- Place the shared data in a cache line aligned and padded cacheable region, and use the CMSIS cache maintenance functions.

  Using a cacheable region requires that the CMSIS cache maintenance functions be used to solve the coherency issues. The data **MUST** be aligned to 32 bytes and be a length multiple of 32 bytes. This is required to meet D-Cache line alignment and length requirements. **If data is not aligned and fit exactly to D-Cache lines, you will create rare and difficult bugs!** Use the CMSIS cache maintenance functions as required to manage the shared data coherency. When the D-Cache is enabled, FSP configures and enables the MPU at startup. Any address outside the predefined MPU regions will use the cacheability attributes defined by the default system address map. **Cacheable shared data cannot be pooled into a single aligned and padded region, like the non-cacheable region. Cacheable shared data must be aligned and padded to its own data cache lines.**

Less common coherency concerns include:

- FACI erasing and programming
  - FCACHE, I-Cache, and D-Cache can become incoherent if Code Flash or Data Flash changes
- Writing instructions into RAM using the CPU or a bus master
  - D-Cache and I-Cache can become incoherent if code in RAM changes
- Memory attributes mismatch between the Secure and Non-secure state when TrustZone is used
  - FCACHE and D-Cache can become incoherent if the cacheability attributes of an address are mismatched between the Secure and Non-secure state
  - Shared addresses must be configured with identical cacheability attributes, either by ensuring that the Secure and Non-secure state use the default system address

map for the shared address, or by configuring the same region in both the Secure and Non-secure MPU with the same attributes
    - I-Cache is usually not affected in this manner, because instruction fetch uses the banked MPU registers which correspond to the address security attribute
    - Consider that complex software designs may swap the current Secure or Non-secure MPU configuration at-will
- MPU configuration changes
    - FCACHE, I-Cache, and D-Cache can become incoherent if the cacheability attributes of an address changes
- SAU (Security Attribution Unit) configuration changes
    - FCACHE, I-Cache, and D-Cache can become incoherent if the security attributes of an address changes
- Changing power modes
    - May be required to clean and invalidate caches before changing to a low power mode

These less common situations require careful consideration for cache maintenance. FSP handles some of these less common situations for FCACHE and I-Cache where it must, but it is not possible to cover all user behavior that may occur.

Other special cache concerns that are not specifically coherency related include:

- Reading and writing from CSC, SDRAM, and Standby SRAM which use write buffers
    - If a write or read is intended to force a write buffer to flush, or to force a bus access to occur, the D-Cache may stop that from occurring by processing the read or write if the address is cacheable
    - Standby SRAM requires a different procedure than CSC or SDRAM to clear its write buffer
- OSPI provides both a prefetch buffer for reading and a write buffer for writing on each of its channels
    - These buffers are optionally enabled individually, and may also be flushed individually
    - Cache interactions with these buffers during normal operation and during their flush procedures must be considered

### 2.3.3 CM85 Cache Features

The CM85 optionally implements an I-Cache and/or D-Cache, with several configurable properties. Renesas RA8 devices have an implementation as follows:

- I-Cache and D-Cache are both implemented with 16 KiB size each
- ECC is optionally enabled for the I-Cache and D-Cache with OFS1.INITECCEN
    - The OFS1_SEL register selects whether the Secure or Non-secure OFS1.INITECCEN option bit is used
    - The OFS1.INITECCEN option bit also enables ECC for the ITCM and DTCM
- Automatic hardware cache invalidation is enabled at reset for I-Cache and D-Cache
    - This can be controlled with CACHEDBGCR.L1RSTDIS for debugging, but generally there is no use case
- TrustZone is integrated, which means
    - CCR.xC is banked between Secure and Non-secure modes
        - Cache allocation can be controlled per the security of the address being fetched, read, or written by configuring the associated bit
        - This bit is safe from causing cache incoherency since allocation policy (allocation hints) cannot cause incoherency, and are not counted towards memory attribute mismatch

        - See the Memory Attributes and TrustZone section
    - There is a Secure and Non-secure MPU with eight regions available for each
        - Which MPU is used depends on whether an instruction fetch (uses MPU that matches security of address) or data access (uses MPU that matches current security state) is occurring
        - See the Memory Attributes and TrustZone section
    - D-Cache maintenance from the Non-secure state is promoted to clean type maintenance since both Secure and Non-secure data may be cached, which prevents Secure data from being modified by the Non-secure state
        - Secure data may still be destroyed by the Non-secure state if it is not properly structured
        - See the Cache Maintenance and TrustZone section

## 2.3.4 RA8 Cache Background Information

### CM85 Caches

The I-Cache and D-Cache are implemented inside of the CM85 by Arm. The CM85 has a Harvard design, where instruction fetches and data reads/writes are performed on separate interfaces. The I-Cache can only perform lookup and allocation for instruction fetches. The D-Cache can only perform lookup and allocation for data reads and writes.

Whether lookup of an address occurs in a cache depends on:

1. Cache lookups are enabled in the MSCR register.
2. The Shareability (Non-shareable) (D-Cache Only), Inner Cacheability (Cacheable), and Memory Type (Normal) as defined by the System Address Map or the Arm MPU (S or NS).
3. Any hardware cacheability caveats, where some addresses can never lookup in a cache, like ITCM and DTCM.

Whether allocation of an address occurs in a cache depends on:

1. Cache allocations are enabled in the CCR register (S or NS).
2. Cache lookups are allowed for that address.
3. Inner Cacheability (Read and/or Write Allocate) as defined by the System Address Map or the Arm MPU (S or NS).

The MSCR register controls whether lookups may occur for a cache, while the CCR register (S or NS) controls whether allocation may occur for a cache.

The system address map and Arm MPU (S or NS) define the memory type, shareability attributes, and cacheability attributes for an address. All three memory properties combine to define whether an address may lookup and allocate within one of the caches. The CM85 defines specific behaviors for some architecturally implementation defined or undefined behaviors regarding combinations of these three properties.

Both caches accept Inner cacheability attributes from the system address map and the Arm MPU (S or NS). The support of cacheability attributes varies depending on the cache and its configuration.

The I-Cache and D-Cache have different associativity, supported cache policies, supported memory attributes, and supported shareability attributes. Because of these variations, the behavior of I-Cache and D-Cache will be different even when accessing the same address.

Even when the Arm MPU (S or NS) is disabled, the default system address map will provide the memory type, shareability attributes, and cacheability attributes for an address. If the Arm MPU (S or

NS) is enabled with no regions defined, it may be configured to use the default system address map as a background region.

Arm prescribes specific procedures for enabling and disabling the caches, and other cache maintenance operations that must be followed. There are I-Cache and D-Cache specific Arm architectural instructions that are used to perform these procedures. The Arm CMSIS library provides functions to perform these cache operations.

**Renesas FCACHE**

The FCACHE is implemented by Renesas and performs instruction prefetches, caches instruction fetches, and caches data reads from the CPU and other bus masters to **Code Flash** memory.

Cache maintenance for FCACHE is conducted through its peripheral registers.

In general, whether for the CM85 I-Cache or D-Cache, or Renesas FCACHE, cache maintenance is used to synchronize a given cache with the backing memory, and to synchronize caches with each other.

## 2.3.5 Cache Maintenance

The correct maintenance sequence must be followed to avoid caches reading stale data from each other or from backing memory.

Unlike the D-Cache, the I-Cache is a read-only interface which cannot be written with new instructions by the CPU. The only way for the CPU to see modified instructions while using I-Cache is through invalidation. Thus if instructions change, I-Cache maintenance is **always required** whether FCACHE maintenance is needed or not.

The D-Cache is a read and write interface. The CPU will write modified data into the D-Cache (if cacheable, and other properties are met), so any read back from the D-Cache will have the latest data. Thus if data changes, it *may* be necessary to perform D-Cache maintenance and possibly FCACHE maintenance depending on whether the data is shared between the CPU and another bus master or if the data is changed by FACI.

The word "shared" here does **not** mean "Shareability" as defined as an Arm memory attribute.

I-Cache and D-Cache cache lines are aligned to 32 bytes and are 32 bytes in length each. **For D-Cache specifically, where a cache line may become dirty when write-back is used, cacheable shared data written by a bus master cannot be allowed to mix on a write-back cache line with data that is unrelated. For simplicity, follow the most conservative rule of aligning and padding cacheable shared data to meet D-Cache line requirements.**

CM85 has an erratum with write-back when D-Cache is enabled. FSP v5.3.0 has added the recommended workaround of using MSCR.FORCEWT to force all D-Cache access to write-through, even if an access specifies write-back. **Developers should write software as if write-back is being used for full compatibility with data cache, which includes the above alignment and padding requirement.**

This guide describes common maintenance scenarios including write-back and write-through. **The write-through write policy does not obviate the need for using memory barriers on the CM85. The use of memory barriers is out-of-scope for this document.**

FSP handles some of these maintenance scenarios during startup and in the Flash HP driver.

## 2.3.6 Typical Cache Maintenance Scenarios

*Note*

***Whether full or address-based cache maintenance can or should be used depends on the capabilities of the target cache (FCACHE has no address-based maintenance capability) and the particular application scenario.***

### I-Cache, FCACHE Enabled (Default Configuration)

### Instructions Change

- Instructions Not Cacheable
  - No Maintenance Required (Not Cacheable)
- Instructions Cacheable
  - FACI Code Flash Program or Erase
    1. Invalidate FCACHE
    2. Invalidate I-Cache
  - FACI Data Flash Program or Erase
    1. Invalidate I-Cache
  - In RAM or Other
    - Written by CPU
      1. Invalidate I-Cache
    - Written by Bus Master
      1. Invalidate I-Cache

### Data Change

- Data Not Cacheable
  - No Maintenance Required (Not Cacheable)
- Data Cacheable
  - FACI Code Flash Program or Erase
    1. Invalidate FCACHE
  - FACI Data Flash Program or Erase
    - No Maintenance Required (D-Cache Disabled)
  - In RAM or Other
    - No Maintenance Required (D-Cache Disabled)

### Instructions and Data Change

- Instructions and Data Not Cacheable
  - No Maintenance Required (Not Cacheable)
- Instructions and Data Cacheable
  - FACI Code Flash Program or Erase
    1. Invalidate FCACHE
    2. Invalidate I-Cache
  - FACI Data Flash Program or Erase
    1. Invalidate I-Cache
  - In RAM or Other
    - Written by CPU
      1. Invalidate I-Cache
    - Written by Bus Master
      1. Invalidate I-Cache

### D-Cache, I-Cache, FCACHE Enabled

### Data Change

- Data Not Cacheable
  - No Maintenance Required (Not Cacheable)
- Data Cacheable
  - FACI Code Flash Program or Erase
    1. Invalidate FCACHE
    2. Clean And Invalidate D-Cache
  - FACI Data Flash Program or Erase
    1. Clean and Invalidate D-Cache
  - In RAM or Other
    - **Data Not Shared**
      - No Maintenance Required (Not Shared)
    - **Data Shared**
      - Written by CPU
        - Write Back

          **Area must be aligned and padded to D-Cache line requirements.**

          1. Clean D-Cache, After CPU Write
        - Write Through
          - No Maintenance Required (Write Through)
      - Written by Bus Master
        - Write Back

          **Area must be aligned and padded to D-Cache line requirements.**

          - Buffer to be Written is Dirty (e.g. Stack or Heap Allocated Buffer May Be Dirty Already)
            1. Invalidate D-Cache, Before and After Bus Master Write
          - Buffer to be Written is Clean
            1. Invalidate D-Cache, After Bus Master Write
        - Write Through
          1. Invalidate D-Cache, After Bus Master Write

**Instructions Change or Instructions and Data Change**

- Instructions Not Cacheable or Instructions and Data Not Cacheable
  - No Maintenance Required (Not Cacheable)
- Instructions Cacheable or Instructions and Data Cacheable
  - FACI Code Flash Program or Erase
    1. Invalidate FCACHE
    2. Clean And Invalidate D-Cache
    3. Invalidate I-Cache
  - FACI Data Flash Program or Erase
    1. Clean And Invalidate D-Cache
    2. Invalidate I-Cache
  - In RAM or Other
    - Written by CPU

- Write Back

  **Area must be aligned and padded to D-Cache line requirements if shared with a bus master.**

  1. Clean D-Cache, After CPU Write
  2. Invalidate I-Cache
- Write Through
  1. Invalidate I-Cache

- Written by Bus Master

  - Write Back

    **Area must be aligned and padded to D-Cache line requirements as we assume it is shared with CPU here.**

    - Buffer to be Written is Dirty (e.g. Stack or Heap Allocated Buffer May Be Dirty Already)
      1. Invalidate D-Cache, Before and After Bus Master Write
      2. Invalidate I-Cache
    - Buffer to be Written is Clean
      1. Invalidate D-Cache, After Bus Master Write
      2. Invalidate I-Cache
  - Write Through
    1. Invalidate D-Cache, After Bus Master Write
    2. Invalidate I-Cache

## 2.3.7 Cache Functions and Macros

### Renesas BSP Configuration Macros

| Macro | Purpose | Notes |
|---|---|---|
| BSP_CFG_DCACHE_ENABLED | Defaults to zero (disabled). If defined and non-zero, the FSP startup code in system.c will configure several predefined non-cacheable sections in the MPU if they are of non-zero size, enable the MPU, and enable the D-Cache. | This is normally configured in e2 Studio under the BSP->Cache settings->Data cache properties panel for the project. |
| BSP_CFG_OPTION_SETTING_OFS1_INITECCEN | Defaults to zero (disabled). Sets the value of OFS1.INITECCEN for BSP_CFG_OPTION_SETTING_OFS1, which controls whether ECC is enabled for caches and TCM. | This is normally configured in e2 Studio under the BSP->OFS1 register settings->Tightly Coupled Memory (TCM)/Cache ECC properties panel for the project. |

### CMSIS 6 I-Cache Functions

| Function | Purpose | Notes |
|---|---|---|

| | | |
|---|---|---|
| SCB_EnableICache | If I-Cache allocations are not already enabled, invalidate the entire I-Cache then enable I-Cache allocations with CCR.IC. | Will do nothing if I-Cache allocations are already enabled. FSP automatically enables the I-Cache at startup by directly setting CCR.IC instead of using this function. |
| SCB_DisableICache | Disable I-Cache allocations with CCR.IC, then invalidate the entire I-Cache. | |
| SCB_InvalidateICache | Invalidate the entire I-Cache. | This is safe to use at any time, because cache lines in the I-Cache can never be dirty. Used after modifying instructions anywhere in memory (e.g. Flash, RAM). FSP calls this function after initializing the predefined RAM code section during startup, and when exiting Code Flash program or erase mode in the Flash HP driver. |
| SCB_InvalidateICache_by_Addr | Loop to invalidate the instructions in the I-Cache, starting at a particular address and extending for the specified length in bytes. | This is safe to use at any time, because cache lines in the I-Cache can never be dirty. Can be used to more efficiently invalidate instructions at specific addresses. Will invalidate in increments of cache lines (32 bytes). |

These functions can be safely interrupted and do not need to be guarded by critical sections. However, depending on the structure of the application logic, guarding the functions may be necessary. This must be analyzed for an individual scenario. See the CMSIS 6 API reference material for further information.

**CMSIS 6 D-Cache Functions**

| Function | Purpose | Notes |
|---|---|---|
| SCB_EnableDCache | If D-Cache allocations are not already enabled, loop to invalidate the entire D-Cache then enable D-Cache allocations with CCR.DC. | Will do nothing if D-Cache allocations are already enabled. FSP automatically calls this function at startup if BSP_CFG_DCACHE_ENABLED is defined and non-zero. |
| SCB_DisableDCache | Disable D-Cache allocations with CCR.DC, then loop to clean and invalidate the entire D-Cache. | |

| | | |
|---|---|---|
| SCB_InvalidateDCache | Loop to invalidate the entire D-Cache. | This function should generally **not** be used, since no use case typically exists to invalidate the entire D-Cache. |
| SCB_CleanDCache | Loop to clean the entire D-Cache. | |
| SCB_CleanInvalidateDCache | Loop to clean and invalidate the entire D-Cache. | |
| SCB_InvalidateDCache_by_Addr | Loop to invalidate the data in the D-Cache, starting at a particular address and extending for the specified length in bytes. | Will invalidate in increments of cache lines (32 bytes). |
| SCB_CleanDCache_by_Addr | Loop to clean the data in the D-Cache, starting at a particular address and extending for the specified length in bytes. | Will clean in increments of cache lines (32 bytes). |
| SCB_CleanInvalidateDCache_by_Addr | Loop to clean and invalidate the data in the D-Cache, starting at a particular address and extending for the specified length in bytes. | Will clean and invalidate in increments of cache lines (32 bytes). |

These functions can be safely interrupted and do not need to be guarded by critical sections. However, depending on the structure of the application logic, guarding the functions may be necessary. This must be analyzed for an individual scenario. See the CMSIS 6 API reference material for further information.

## 2.3.8 Cache Details

### I-Cache Details

### FSP I-Cache and FCACHE Behavior

Because of the simplicity of the I-Cache and FCACHE relative to the D-Cache and the critical instruction execution performance enhancement that they provide, FSP always enables the I-Cache and the FCACHE. This is not configurable.

FSP automatically enables the I-Cache at startup for CM85 by directly setting CCR.IC. This method is used instead of the CMSIS 6 function, so that the I-Cache, branch prediction, and the low-overhead branch (LOB) extension may simultaneously be enabled. The automatic hardware cache invalidation of the CM85 ensures that cache lookups, allocations, and cache maintenance are no-op until the invalidation is finished, so immediately enabling the I-Cache is safe to do.

FSP invalidates the I-Cache using the CMSIS 6 functions:

- After initialzing the predefined RAM code section during startup
- After initializing the SAU for a Secure TZ application
- When exiting Code Flash program or erase mode in the Flash HP driver

The FCACHE is a Renesas cache, not an Arm cache, and it is controlled through its separate peripheral registers.

### User Required I-Cache Maintenance

If instructions have changed outside of the control of FSP, it is user responsibility to perform I-Cache maintenance. This means instructions stored in *any* cacheable location, including internal flash, internal RAM, external flash, external RAM, etc. It is recommended to use the CMSIS 6 functions to perform I-Cache maintenance.

Users must also consider the interactions that D-Cache has with instruction modifications. For example, if the modified instructions are written to a cacheable location while D-Cache is enabled (e.g. RAM), those data writes may be cached. The D-Cache will need to be cleaned to guarantee that all data writes have been written back to guarantee their visibility to the I-Cache. D-Cache maintenance will also be needed if instructions change in Code or Data Flash via FACI and are cacheable, since D-Cache will cache instructions as data.

There is no hardware mechanism between the I-Cache and D-Cache in which they automatically share coherency, so coherency must be manually maintained by software as required.

### D-Cache Details

### FSP D-Cache Behavior

The D-Cache is a cache with more complex interactions than the I-Cache. Thus, FSP leaves the D-Cache disabled by default on RA8 projects. It can be enabled in e2 Studio under the BSP->Cache settings->Data cache properties panel for the project.

Presently, FSP does not support any D-Cache functionality except:

- Configuring the Arm MPU (S and NS) with two predefined no-cache sections if they are non-zero size
    - One in SRAM
    - One in SDRAM
- Enabling the Arm MPU (S and NS) after configuration
- Enabling D-Cache allocations (S or NS) with the CMSIS function

**No FSP drivers are currently compatible with D-Cache enablement. This compatibility is a work in-progress.**

### User Required D-Cache Maintenance

Presently, D-Cache usage is fully in the realm of user responsibility. The user must perform all D-Cache maintenance as required, or must store data accordingly in the predefined non-cacheable regions or otherwise.

### Affected Bus Masters

Coherency must be considered for these bus masters:

- CEU
- DMAC
- DRW
- DTC
- EDMAC
- GLCDC
- MIPI

#### Other Interactions that are not Bus Masters

Coherency must be considered for interactions with:

- FACI
- CSC
- SDRAM
- Standby SRAM
- OSPI
- Code in RAM (I-Cache and D-Cache are not automatically coherent)

#### FSP Predefined No-Cache Sections

Warning

**The predefined non-cacheable sections are only understood as non-cacheable within their respective security state when TrustZone is used. This is due to the MPU configuration done by FSP and how Armv8-M chooses the MPU which matches the current security state for determining the memory attributes of a data access. Cache incoherency will occur if references to these sections are passed between security states. For example, passing a reference from the Non-secure application non-cacheable section to the Secure application will result in the Secure application treating the location as cacheable.**
**The predefined non-cacheable sections are only understood as non-cacheable within their respective core. In dual-core devices, each core has a separate MPU which may configure its own regions and attributes, so these sections cannot be used for coherent interprocessor communications in the case that D-Cache is enabled. Use the IPC peripheral instead.**

Several non-cacheable linker sections are predefined for GCC, LLVM, and IAR compilers, which may be zero-initialized or uninitialized. These same sections exist for AC6 prefixed with .bss, due to special naming restrictions with AC6 for non-loadable sections.

Anything placed within them will be non-cacheable. Instruction fetches and data reads or writes to these sections will never lookup or allocate in their respective caches.

The FSP startup code configures these sections as non-cacheable using the MPU during startup, if the D-Cache is enabled via the BSP configuration. Otherwise, they are not configured by FSP in the MPU if the D-Cache is disabled. The predefined sections are aligned to 32 bytes and are padded to a minimum of 32 bytes in length. This meets both MPU region alignment and length requirements, and cache line alignment and length requirements. The MPU and cache line alignment and length requirements protect against inadvertent mixing of cacheable and non-cacheable data.

### 2.3.9 Other Information

#### Cache Maintenance when MPU Configuration Changes

If the Secure and/or Non-secure MPU configuration is changed, and the cacheability of an address changes, cache maintenance is required to synchronize the caches with the new memory attributes. If this is not done, a newly non-cacheable address may be left in the cache, and behavior when accessing the address is considered **undefined**.

#### Cache Maintenance when SAU Configuration Changes

If the SAU configuration is changed, and the security attributes of an address changes, cache maintenance is required to synchronize the caches with the new security attributes. If this is not

done, cached data will be desynchronized with the new security attributes and may result in **undefined** behavior.

### Memory Attributes and TrustZone

Warning

**If data that is shared between the Secure and Non-secure application does not use the same memory attributes while being accessed because of a memory attribute mismatch, it will cause cache incoherency.** See the Arm knowledgebase article ka001216 and the Armv8-M Architecture Reference Manual for information on MPU selection behavior and memory attribute mismatch.

### Cache Maintenance and TrustZone

Warning

**If cacheable shared data is improperly structured by the Secure application and is not aligned and padded to match D-Cache line requirements, a clean and invalidate by set/way of the D-Cache by the Non-secure application, or an automatic D-Cache eviction by the Non-secure application, will cause data to be destroyed for the Secure application.** This consequence is additional to the bugs that the Secure application may trigger itself by improper data layout, D-Cache maintenance, and automatic D-Cache eviction.

### The System Address Map and the MPU

The Armv8-M Architecture Reference Manual specifies a default system address map that defines the memory regions of the architecture and their various properties.

These properties include the memory type, shareability attributes, and cacheability attributes for the regions. When the MPU is disabled, this default system address map provides the system with default attributes for instruction fetches, data reads, and data writes to and from addresses. When the MPU is enabled, it can be used to override the default system address map entirely, or both may be used together by setting the MPU_CTRL.PRIVDEFENA bit. This bit allows instruction fetches or data reads and writes that do not correspond to a configured MPU region to hit the default system address map as a background region instead, so long as the access is Privileged. FSP does not support Unprivileged execution, so it always assumes Privileged execution state. Allowing the default system address map as a background region is the method that FSP uses to provide the predefined no-cache sections, by configuring the MPU for the no-cache sections while allowing all other memory accesses to rely on the default system address map. Configuring an MPU region involves specifying a 32 byte aligned start address and an inclusive ending address, and also specifying the various memory attributes of the region. The MPU region beginning address register will mask downward to align to a 32 byte boundary. i.e (address & ~0x1F) The MPU region ending address register will OR upward with 0x1F for the inclusive ending boundary. i.e. (address | 0x1F) Thus, the minimum size of an MPU region is 32 bytes and the size may only increase in 32 byte increments.

See the Armv8-M Memory Model and Memory Protection User Guide in the references section for a high-level introduction, and the Armv8-M Architecture Reference Manual for details.

### Speculative Instruction Fetching and Data Reads

The CM85 may, with no deliberate software instruction, speculatively fetch instructions or read data from any memory location. Upon doing so, the instruction fetch or data read may enter the respective cache. The purpose of this speculative behavior is to predict the next instructions or data to be fetched, read, or written, which increases performance if the prediction is correct. **This may cause instructions or data to unexpectedly appear in cache, so speculation must be**

**considered when solving for cache coherency.**

### Cache Eviction

At any time, the I-Cache or D-Cache may evict cache lines. For I-Cache, this means invalidation of the evicted line. For D-Cache, this means cleaning and invalidation of the evicted line, where cleaning occurs if the cache line is dirty. **D-Cache eviction of dirty lines may cause data to be unexpectedly written out to backing memory when write-back is used, and this must be considered when solving for cache coherency. For D-Cache aligned and padded buffers derived from areas like the stack or heap, one or more of the associated cache lines may already be dirty and require cleaning and/or invalidation before being used by a bus master.**

### Example of D-Cache Eviction and Speculative Read Dangers

The CM85 will provide a SRAM buffer to the DMAC, the DMAC will write to the buffer, and the CM85 will read from the written buffer. I-Cache, D-Cache, and FCACHE are enabled. SRAM exists in the same "SRAM" region defined by the default system address map in the Armv8-M Architecture Reference Manual. SRAM is Normal memory, write-back, write-allocate, read-allocate, and non-shareable by the default system address map attributes.

The correct way to solve coherency in this situation using the two recommended solution options is:

1. The MPU is used to configure a non-cacheable region where the SRAM buffer is placed.
    - The MPU region is correctly aligned and padded to meet start and end address alignment requirements, and region attributes are correctly configured.
    - No additional effort is required.
    - FSP provides predefined non-cacheable sections that meet these requirements.
2. The SRAM buffer is left to its default attributes, making it cacheable.
    - Regardless of whether write-back or write-through is used, the buffer start must be aligned to a cache line and the buffer must be a length multiple of cache line size.
        - Unrelated data can never be mixed on cache lines if write-back is used. It is best to follow this strict guideline even when write-through is used.
    - The cache lines of the buffer may already be dirty, especially if the buffer is allocated in a stack or heap.
    - If write-back is used, cache maintenance is conducted in order.
        - Invalidate buffer.
        - CM85 provides buffer to DMAC and starts DMAC.
        - DMAC writes to buffer.
        - CM85 waits for DMAC to complete.
        - Invalidate buffer.
        - CM85 reads from buffer.
    - The first invalidation is to remove dirty lines, which may already exist from stack or heap allocation. The data does not need to be written back and can be discarded without a clean.
        - If this is not done, an eviction (effectively an automatic clean and invalidate by the hardware) will cause stale data to be written back to the buffer and destroy newly written DMAC data.
    - The second invalidation is to remove speculatively read cache lines, which may have been cached before the DMAC write completed.
        - If this is not done, the CM85 will read stale data from the buffer that has been prematurely cached.
    - Write-through may also use this sequence, although the first invalidate should be a no-op since no lines can be dirty and no stale data should be written back by an eviction.

- The CMSIS cache maintenance functions should be used to perform cache maintenance, since they include necessary memory barriers.

The second solution is shown in the D-Cache Enabled scenarios here where write-back is used and a bus master performs writing.

#### 2.3.9.1 Cache ECC with FSP

By default, FSP disables ECC for the caches and TCM with OFS1.INITECCEN. For best performance, it is recommended to keep ECC for cache and TCM disabled. If enabling is desired, please consult the reference material to understand the consequences of enabling ECC for cache and TCM, which are too numerous to describe here. Also see Devices with OFS1.INITECCEN. The automatic hardware cache invalidation performed by the CM85 is compatible with ECC.

**MPU Cacheability Attributes**

- Cacheable or Non-Cacheable
- Allocation Policies
  - Read Allocate
  - Write Allocate
- Write Policy
  - Write Back or Write Through
- Transient or Non-Transient

For D-Cache, Shareable or Non-Shareable also affects whether an address is Cacheable or Non-Cacheable. A Shareable address is forced to Non-Cacheable for D-Cache. I-Cache is not influenced by the Shareability properties and will always follow the MPU cacheability attributes. The Transient attribute is of limited utility and can mostly be ignored. Clean cache lines that are marked Transient are preferred for eviction before clean cache lines marked Non-Transient. Dirty cache lines whether marked Transient or Non-Transient are evicted with the same priority.

See the CM85 Technical Reference Manual reference material for further information.

**Cache Behavior with CCR and MSCR**

x = [I, D]

| CCR.xC (S or NS) | MSCR.xCACTIVE | Behavior |
|---|---|---|
| 1 | 1 | Allocate, Lookup |
| 0 | 1 | No Allocate, Lookup (Reset Behavior) |
| X | 0 | No Allocate, No Lookup |

This behavior is applicable to Cortex-M55 and Cortex-M85. If you have previous experience with a Cortex-M7 device, this cache behavior is different since MSCR.xCACTIVE bits were introduced for CM55 and CM85. No CM7 or CM55 core is offered by any current RA devices. The new addition of the MSCR.xCACTIVE bits allow for cache power control, and by allowing a third cache behavioral state of lookups without allocation, cleaning the D-Cache after disabling it becomes less error prone since dirty cache lines cannot be made stale before being cleaned, by writes occurring after D-Cache is disabled like on CM7. The MSCR.xCACTIVE bits have a reset value of 1, so the caches are powered by default and lookups are possible. Until the automatic hardware cache invalidation which begins after reset finishes, lookups and allocations do not occur even if CCR.xC is set, and cache maintenance

operations are no-op. The MSCR.xCACTIVE bits should generally never be cleared to 0.

### Cache Errata

Consult the latest Renesas Technical Updates (TU) and Arm Cortex-M85 Errata documents.

These are example errata to demonstrate the possibility of issues with cache usage at the time of this writing.

```
Cortex-M85 AT640 and Cortex-M85 with FPU AT641
Software Developer Errata Notice
Date of issue: April 16, 2024
Document version: 14.0
Document ID: SDEN-2236668
2682779
After deactivating the instruction cache, self-modified code might not be executed
correctly
Fault Type: Programmer Category C
Fault Status: Present in r0p0, r0p1, r0p2. Fixed in r1p0
3175626
AXI hang due to dependency between read data channel and write response channel
Fault Type: Programmer Category B
Fault Status: Present in r0p0, r0p1, r0p2, r1p0. Fixed in r1p1
3190818
Under limited circumstances, LDM to normal non-cacheable AXI location cannot complete
Fault Type: Programmer Category B
Fault Status: Present in r0p0, r0p1, r0p2, and r1p0. Fixed in r1p1
```

Currently available RA8D1, RA8M1, and RA8T1 devices use the r0p2 variant of the core, so they are affected by these errata.

Erratum 2682779 should not require a workaround, since I-Cache will never be powered off in most circumstances.

FSP added workarounds for 3175626 and 3190818 in v5.3.0.

## 2.3.10 References

*Note*

*Cross-reference documents from multiple sources and consult with colleagues and other support channels for maximum confidence.*

### Renesas

Generally, consult these categories of documents for the most recent and further information than this overview may provide.

- RA Datasheets
- RA Hardware User Manuals (HWM, HWUM, UM)
- RA Application Notes (AN)
- RA Knowledge Base Articles (KB)
- RA Technical Updates (TU)
- RA Example Projects

### RA8D1

1. RA8D1 Product Page
2. RA8D1 Datasheet
3. RA8D1 User's Manual: Hardware

### RA8M1

1. RA8M1 Product Page
2. RA8M1 Datasheet
3. RA8M1 User's Manual: Hardware

### RA8T1

1. RA8T1 Product Page
2. RA8T1 Datasheet
3. RA8T1 User's Manual: Hardware

### BSP Usage Notes

1. Limited D-Cache Support
2. Non-Cacheable Buffer Placement Example

### Arm

*Note*

***Arm links appended with "latest" may not actually resolve to the most recent document, because of issues with the Arm documentation website. Always check that the document you are accessing is truly the most recent version using the version drop-down list box.***

### Armv8-M and Armv8.1-M Architectures

1. Armv8-M Architecture Reference Manual
2. Armv8-M Memory Model and Memory Protection User Guide
3. Armv8-M Exception Model User Guide
4. Which MPU configuration is used in a Trustzone enabled Armv8-M system (ka001216)

### Cortex-M85

1. Cortex-M85 Product Page
2. Arm Cortex-M85 Processor Technical Reference Manual
3. Arm Cortex-M85 Processor Devices Generic User Guide
4. Cortex-M85 AT640 and Cortex-M85 with FPU AT641 Software Developer Errata Notice

5. Arm Cortex-M85 Processor Software Optimization Guide

**CMSIS 6**

1. CMSIS 6 GitHub Repository
2. CMSIS 6 Documentation
3. CMSIS 6 MPU API for Armv8-M
4. CMSIS 6 Cache API

# 2.4 Using Ocrypto

## 2.4.1 Overview

Ocrypto is a lightweight software crypto implementation by Oberon.

## 2.4.2 Limitations

The Ocrypto library is only to be used with RA2E1 devices.

## 2.4.3 RA FSP Licensed Algorithms

Below is a table that lists the Ocrypto algorithms licensed for use with RA FSP. Consider this table when referencing the below section - Additional Information

| Algorithm | API Header Files |
|---|---|
| SHA (224, 256, 384, 512) | ocrypto_sha224.h<br>ocrypto_sha256.h<br>ocrypto_sha384.h<br>ocrypto_sha512.h |
| HMAC (SHA256, SHA512) | ocrypto_hmac_sha256.h<br>ocrypto_hmac_sha512.h |
| AES-CMAC | ocrypto_aes_cmac.h |
| AES (CTR, CBC, ECB, GCM) | ocrypto_aes_cbc.h<br>ocrypto_aes_cbc_pkcs.h<br>ocrypto_aes_ctr.h<br>ocrypto_aes_ecb.h<br>ocrypto_aes_gcm.h<br>ocrypto_aes_key.h |
| ECDSA / Deterministic ECDSA (P256) | ocrypto_ecdsa_p256.h<br>ocrypto_ecdsa_p256_det_sign.h |
| ECDH (P256) | ocrypto_ecdh_p256.h |

## 2.4.4 API Examples

### Ocrypto Library Examples

For examples of how to use Ocrypto APIs, see Ocrypto test code under ra/oberon/ocrypto/tests/. These tests are not meant to be built or run in FSP and have been compiled out by the macro OCRYPTO_REFERENCE_TEST_CODE_DOES_NOT_BUILD by default.

## 2.4.5 Additional Information

| Document | Summary |
|---|---|
| Ocrypto - Side Channel Safety Statement | Side Channel Safety Statement for the Oberon Ocrypto stack |
| Ocrypto - Function Overview | Function Overview for the Oberon Ocrypto stack |

## 2.4.6 Ocrypto - Side Channel Safety Statement

# Side-Channel Safety Statement

Copyright Oberon microsystems AG (since 2019)

### 1) Timing attacks, simple power/radiation analysis attacks

These attacks rely on data-dependent timing variations in the used algorithms. They can be applied passively, just by observing normal operation.

All *ocrypto* functions (see chapter 5) are PC-secure (execution paths do not depend on input data) and constant-time (execution time is independent of input data) except for the list of functions with special behavior listed below (see chapter 4).

A special case of timing variations are table reads with data-dependent indices. Accessing data-dependent memory addresses may influence the execution timing because of address-specific cache reload operations. Data-dependent table lookups may be dangerous even on systems without data cache because the table lookups in RAM or flash memory may dominate chip radiation if other calculations are done mainly in registers.

All *ocrypto* functions are free of data-dependent table reads except for the functions with special behavior listed below (see chapter 4).

### 2) Differential power/radiation analysis attacks

These attacks improve the precision of the analysis by averaging over several (typically hundreds of) runs and/or comparing several runs by feeding specific input data. They can only be applied if the system permits an arbitrary number of tries with either the same or arbitrarily chosen input data. Whether and what kind of countermeasures are needed for avoiding such attacks largely depends on the usage of the functions, not the functions themselves.

*ocrypto* does not contain any special provisions against differential attacks.

**3) Fault attacks**

In addition to differential analysis, fault attacks rely on actively influencing the attacked system to provoke faults in the execution. Countermeasures include physical protection of the device and/or attack detection mechanisms on all levels. Detection on an algorithm level is usually expensive and only justified if fault attacks are actually feasible for a given system.

*ocrypto* does not contain any special provisions against fault attacks.

**4) List of ocrypto functions with special behavior**

This list shows functions with special behavior:

- Ed25519/Ed25519ph/Ed448/Ed448ph verify

  Ed25519 et.al. verify is not constant-time "by design". The algorithm is chosen such that a highly efficient variable-time double multiplication algorithm can be used. This is justified by the fact that signature verification depends on public values only (signed data, public signature key). The variable-time multiplication does not harm even if the data or the public key are considered to be secret, because neither of them is directly involved in the multiplication, only values derived from them using secure hash functions.

- Ed25519/Ed25519/Ed448/Ed448ph public key and sign with "#define OCRYPTO_NO_CACHE"

  The point multiplication uses a table of base point multiples. Without caches, this poses no problems, but see the initial remark on table-based algorithms (chapter 1).

- RSA with "#define OCRYPTO_NO_CACHE"

  The RSA exponentiation uses a table of small exponents of the argument. Without caches, this may pose no problems, but see the initial remark on table-based algorithms (chapter 1).

- SRP

  The SRP calculations are not perfectly constant-time, because some hash values are taken on a compact representation of some critical values. In the worst case, this can reveal the number of leading zero bytes in these values, including the premaster secret. This cannot be avoided and is considered a design error in the SRP specification.

- ML-DSA and ML-KEM

  ML-DSA and ML-KEM are not constant-time because of rejection sampling in several places. However, the timing can only disclose information on values that are not used.

  ML-DSA is not table-free because the SampleInBall algorithm permutes array elements with pseudo-randomly derived indices. However, the seed of the resulting polynomial is part of the signature and thus public anyway.

**5) List of ocrypto functions**

This is the list of functions supported by *ocrypto*. Note that your specific distribution of *ocrypto* may

contain just a subset of these functions.

All algorithms except for Ed25519/Ed25519ph/Ed448/Ed448ph are implemented in a PC-secure way (i.e., always the same sequence of instructions is executed).

| Algorithm | Const-Time | Tables |
|---|---|---|
| SHA | yes | safe |
| SHA2 (224, 256, 384, 512) | yes | safe |
| SHA3 (224, 256, 384, 512) | yes | safe |
| SHAKE / cSHAKE (128, 256) | yes | safe |
| Ascon-Hash256 | yes | safe |
| HMAC (SHA1, SHA256, SHA512) | yes | safe |
| AES-CMAC | yes | safe |
| KMAC (128, 256) | yes | safe |
| ChaCha20 | yes | safe |
| XChaCha20 | yes | safe |
| AES (CTR, ECB, CBC) | yes | safe |
| AES (CCM, GCM, EAX) | yes | safe |
| ChaCha20-Poly1305 | yes | safe |
| XChaCha20-Poly1305 | yes | safe |
| Ascon-AEAD128 | yes | safe |
| HKDF (SHA1, SHA256, SHA512) | yes | safe |
| PBKDF2-HMAC (SHA1, SHA256) | yes | safe |
| PBKDF2-AES-CMAC-PRF128 | yes | safe |
| RSA (PKCS1, PSS) | yes | * |
| ECDSA (P-224, P-256, P-384, P-521) | yes | safe |
| ECDSA secp256k1 | yes | safe |
| EdDSA / EdDSAph (Ed25519, Ed448) | * | * |
| LMS/HSS verify (SHA256/SHAKE256) | yes | safe |
| XMSS/XMSS^MT verify (SHA256/SHAKE256) | yes | safe |
| ML-DSA (44, 56, 87) | * | * |
| RSA (PKCS1, OAEP) | yes | * |

| | | |
|---|---|---|
| ECDH (P-224, P-256, P-384, P-521) | yes | safe |
| ECDH secp256k1 | yes | safe |
| ECDH (X25519, X448) | yes | safe |
| EC-JPAKE (P-256) | yes | safe |
| SPAKE2+ (P-256, HMAC) | yes | safe |
| SRP-6 (SHA512) | * | safe |
| ML-KEM (512, 768, 1024) | * | safe |

* See comments in chapter 4.

*Note: If you build for the pure C Generic platform, make sure that your compiler issues constant-time multiply operations.*

### 2.4.7 Ocrypto - Function Overview

# Function Overview

**Algorithms**

This version of *ocrypto* supports the following algorithms:

| Message digests (hashes) | Specification |
|---|---|
| SHA2 (224, 256, 384, 512) | FIPS-PUB-180-4 |
| Message authentication codes (MAC) | Specification |
| HMAC (SHA256, SHA512) | RFC2104 |
| AES-CMAC | RFC4493, NIST SP 800-38B |
| Unauthenticated ciphers | Specification |
| AES (CTR, ECB, CBC) | FIPS-PUB-197, NIST SP 800-38* |
| Authenticated encryption with associated data (AEAD) | Specification |
| AES (GCM) | FIPS-PUB-197, NIST SP 800-38* |
| Asymmetric signature | Specification |
| ECDSA / Deterministic ECDSA (P-256) | FIPS-PUB-186-5 |
| Key agreement | Specification |

| | |
|---|---|
| ECDH (P-256) | NIST SP 800-56A |

### Key Sizes

- AES: 128, 192, and 256 bit keys are supported for all uses of AES.
- NIST P-256 curve (aka secp256r1) supports 256 bit keys.

### Function Types

*ocrypto* supports these two types of functions:

### Single-part

Function is executed in a single call, using only the stack.

### Incremental

Function is executed in multiple calls, to allow for streams of data to be processed in a piecemeal fashion. Additional memory is provided by caller in a context object. The functions are split into sets of functions whose names end in init, final and update.

### Function Type Matrix

This table shows which function types are supported for which algorithms:

| Algorithm | Single-part | Incremental |
|---|---|---|
| SHA | x | x |
| HMAC | x | x |
| AES-CMAC | x | |
| AES | x | x |
| ECDSA / Deterministic ECDSA (NIST curves) | x | |
| ECDH (NIST curves) | x | |

There are no stack-optimized functions in this distribution of *ocrypto*.

All functions are available as single-part function types.

### RAM Requirements

*ocrypto* does not use static memory (no global variables).

*ocrypto* does not use dynamic memory allocation - no heap is required.

The following table indicates how much memory is used on the stack:

| Function and type | Required size (bytes) |
|---|---|
| Single-part | < 1,000 |
| Incremental | < 300 |

For incremental functions, the caller must provide pre-allocated context object, of similar size as stack space for single-part functions. The application may allocate these context objects statically or in a heap.

# 2.5 Linker Script Generation Guide

## 2.5.1 FSP Linker Scripts

FSP prior to v6.0.0 provided linker scripts that were common to all MCUs. Only having a single linker script per toolchain had become increasingly complex as the RA MCU family continued to grow and diversify. Starting in FSP v6.0.0, linker scripts are specific to the toolchain, MCU, and type of project being built. This means that users will need to migrate any existing linker script modifications that might have been made when migrating to FSP v6.0.0 or later.

In this guide, we will examine the methodology behind the FSP6 linker script mechanism, highlighting key areas that have changed from earlier FSP releases and providing details of how to migrate user-modified linker scripts from earlier releases to the new mechanism.

> **Audience:** customers wishing deeper knowledge of the new linker scripting or migrating existing Renesas RA projects utilizing BSP_PLACE_IN_SECTION or custom linker scripts to the policy-generated linker scripts.

> **Note:** When using a Solution project it is recommended to use the Memories tab in the configuration editor to modify memory region sizes and location as necessary.

---

### 2.5.1.1 Why we changed

Hand-edited linker scripts had become **non-maintainable** as devices and project styles grew more complex:

- Multiple projects per MCU (bootloader, TrustZone, multicore), each with different **memory partitions**
- Device-specific **capabilities** (OFS/BPS layouts, ITCM/DTCM, SDRAM, OSPI, etc.)
- Divergent behavior across **toolchains** (GCC/LLVM/IAR/ArmCC6)
- Customer requests for **non-invasive placement** (no source edits) and **multi-RAM init**

**What the new system delivers**

- **One policy for four toolchains**: scripts are **generated** from a common policy so behavior is equivalent across GCC, LLVM/Clang, IAR, and AC6.

- **Per-device awareness**: OFS register locations and available memories (ITCM/DTCM/SDRAM/OSPI) are baked in at generation time.
- **Non-invasive placement**: place variables/functions/sections into arbitrary memories **without touching source** via the DDSC **Linker Section Mapping** dialog.
- **Multi-RAM initialization**: support for multiple RAM families (internal RAM, TCMs, SDRAM) with correct copy/zero at startup.
- **Solution-level partitioning**: allocate FLASH/RAM across projects in a **Solution Partitioning** dialog; scripts follow those partitions automatically.
- **Fine-grained OFS Registers**: separate, explicit sections for each register region instead of one all-or-nothing block.

---

**What changed (layout & file model)**

Each toolchain now uses the same two-file include pattern:

- **Region Information** memory_regions.* – *generated*. Declares device/solution memory bases & lengths (e.g., FLASH/RAM/ITCM/DTCM/SDRAM/OSPI, OFS/OTP, etc.). **Do not edit**; update via the Solution Partitioning dialog.
- **Linker Script** fsp_gen.* – *generated*. Defines output sections and default mappings for the current device and toolchain. **Do not edit**; change behavior with mapping rules.
- **Top-level** fsp.* – a tiny wrapper that INCLUDEs the two files above. This file is stable and meant to be referenced by the project.

  If you need custom script logic, prefer **mapping rules** first. If absolutely necessary, either: 1) keep a **patched copy** of the generated files (applied during your build before link), or 2) **fork** to new filenames and INCLUDE them from fsp.* (works, but you **disconnect** from tool-assisted mapping/partitioning).

---

### 2.5.1.2 New section model (high level)

Instead of monolithic .text/.data/.bss outputs, the new scripts split into **semantically named outputs** per memory and init behavior. These names are **consistent across toolchains** and expose **Base/Limit symbols** per region.

Common outputs (examples; presence depends on device/memory availability):

- **FLASH**
  - __flash_vectors$$ – Vector table / Reset handler
  - __flash_readonly_gap$$ – On devices with OFS registers at 0x400, this is space before that space, where we default map content to optimize memory utilization via the DDSC linker section mapping
  - __flash_readonly$$ – Code & const data (also GOT/EXIDX/ctor arrays: __flash_.got$$, __flash_arm.exidx$$, __flash_{preinit,init,fini}_array$$)
  - __flash_noinit$$ – Reserved/no-load content in flash
- **Internal RAM**
  - __ram_dtc_vector$$ - The vector table for the dtc peripheral (located at start of ram for alignment restrictions)
  - __ram_from_{ROM REGION}$$ – Initialized data copied from each available persistent source

    - __ram_zero$$ – Zero-initialized bss/tbss
    - __ram_noinit$$ – Non-initialized (e.g., heaps, main stack)
    - Thread/ELF TLS helpers: __ram_tdata$$, __ram_tbss$$, __ram_thread_stack$$
    - Optional **no-cache** variants (device-dependent): __ram_zero_nocache$$, __ram_noinit_nocache$$
- **Secondary ROM Memories** (such as data flash, qspi, ospi)
    - __{ROM Region}_readonly$$ - Initialized space in this region (constants, code)
    - __{ROM Region}_noinit$$ - Uninitialized space in this region that application code can write at runtime
- **Secondary RAM Memories** (such as tcm, ospi, sdram)
    - __{RAM Region}_from_{ROM Region}$$ - Initialized data copied from each available persistent source
    - __{RAM Region}_zero$$ – Zero-initialized data in this region
    - __{RAM Region}_noinit$$ – Non-initialized data in this region
- **OFS Registers** (fine-grained)
    - __option_setting_{OFS Register}_reg$$ - Fine grained output section for each OFS register

Each output exposes ...$$Base / ...$$Limit symbols for portable startup code and diagnostics across toolchains.

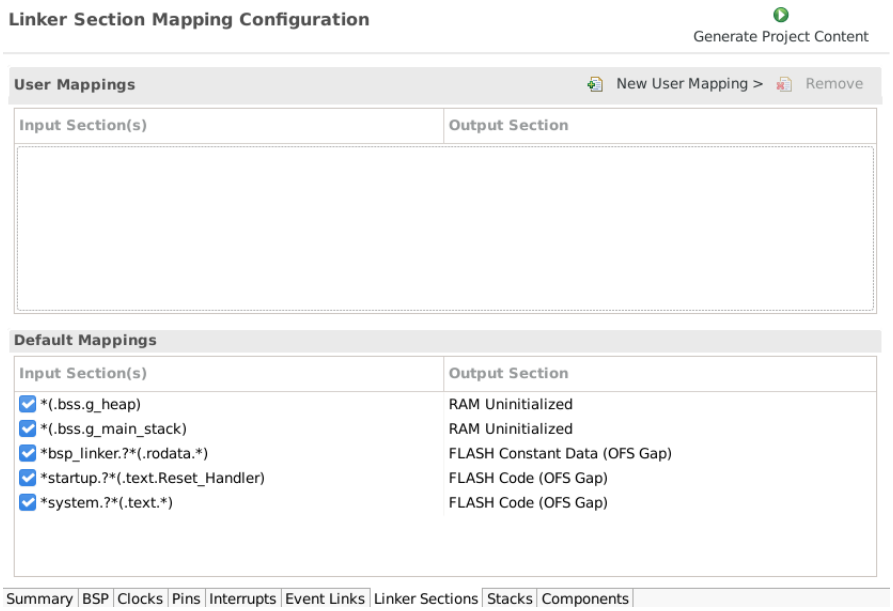
#### 2.5.1.3 Interacting with the new flow

**A) Place objects or code without source edits (recommended)**

Use **DDSC → Linker Section Mapping**

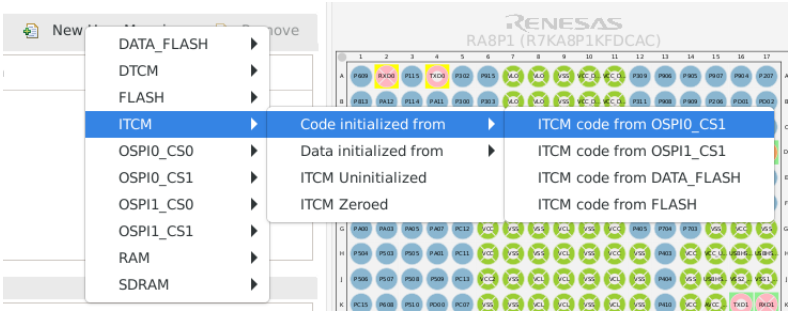
Here you can see typical default mappings for a device with the OFS flash gap.

- The heap and stack are mapped to uninitialized RAM (otherwise they would be placed in zeroed RAM memory)
    - Unchecking the default selection will remove the mapping and allow standard linker placement.
    - These allocations can be overridden to other regions. For example the stack could be placed in DTCM by unchecking the selection and creating a new input section mapping.
- Additionally, we are default placing content into the flash gap to maximize flash utilization.

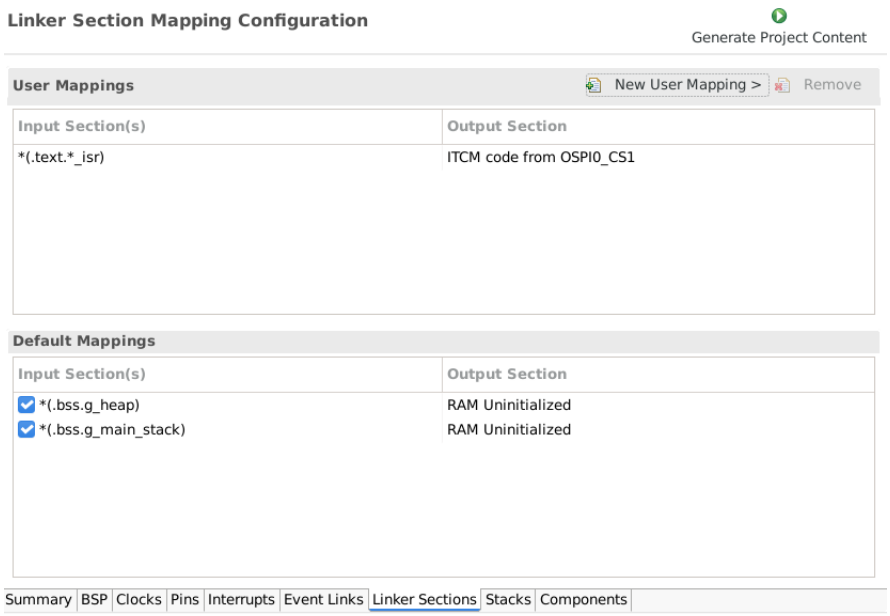



To create a new user mapping select the desired destination (and the intialization source if appropriate)

- Here we are selecting ITCM Code initialized from ITCM code from OSPI0_CS1
- In the dialog that is presented, we enter *(.text.*_isr)
- The syntax of these mappings follow gnu ld input section selection rules (including wildcard patterns) regardless of the toolchain




This means any function that ends in _isr will be persistently stored in OSPI0_CS1 space (in the output elf and srecord) and loaded to ITCM by the fsp startup code.




**B) Source-level placement (still supported)**

BSP_PLACE_IN_SECTION("<name>") continues to work. Some legacy section names changed for consistency; **prefer mapping rules** when migrating. If you keep the macro, use the **input names** as defined in the appendix.

**C) Solution-level partitioning**

Use **Solution Partitioning** to assign FLASH/RAM ranges across bootloader/app/TZ projects. memory_regions.* is regenerated to reflect those partitions; all scripts honor them automatically. Please refer to solution project documentation for details.

---

### 2.5.1.4 Migration guide (step-by-step)

1. **Update** to FSP 6.0 or later that includes the new behavior.
2. **Reference any customer edits** made to the legacy linker scripts and determine how to apply to the generated scripts.
3. **Review defaults** in **Linker Section Mapping**. Add rules for any special placements you previously had in hand-edited scripts.
4. If you used BSP_PLACE_IN_SECTION, either **remove** it and add a mapping rule, or **rename** the section to the new input names shown below.
5. **Build & verify** the project using the new scripts and compare the map files to prior.
6. Only if needed, implement a **patch step** (or a **fork + INCLUDE**) to carry forward any exotic script logic.

---

### 2.5.1.5 Legacy (gcc, llvm) → New **output** section translation

```
legacy_to_new_output_sections:
  .text: [__flash_vectors$$ (Reset handler & vectors), __flash_readonly$$
(code/rodata)]
```

```
  .data: __ram_from_flash$$
  .bss: __ram_zero$$
  .noinit: __ram_noinit$$
  .tdata: __ram_tdata$$
  .tbss: __ram_tbss$$
  .data_flash: [__data_flash_readonly$$, __data_flash_noinit$$]
  .qspi_flash/.OSPI_DEVICE_0/.OSPI_DEVICE_1: OSPI family outputs #(e.g.,
__ospi0_cs0_from_flash$$, __ospi0_cs1_readonly$$) – pick readonly/from/zero/noinit as
needed
  .got: __flash_.got$$
  .ARM.exidx/.ARM.extab: __flash_arm.exidx$$
  .option_setting_* (old single blob): fine?grained __option_setting_*_reg$$
```

**Note:** In the e2 Studio tooling, dialogs related to obj-copy do not properly parse the $$ suffix, so use globbing if required (such as --remove-section=__option_setting_*)

**Note:** exact availability of *_from_* and *_nocache outputs depends on the device and selected memories.

---

### 2.5.1.6 Behavior differences worth noting

- **Generated files overwrite**: fsp_gen.* and memory_regions.* are regenerated. Put customizations in **mapping rules** or in **patches/forks**.
- **Vectors moved**: Reset/ISR vectors live in __flash_vectors$$. General code/rodata live in __flash_readonly$$ (placed after **OFS flash gap** where required).
- **Copy/zero tables generalized**: initialization now spans multiple memories (RAM/TCM/SDRAM/OSPI) with per-output Base/Limit symbols.
- **Fine-grained protection**: OFS/BPS/OTP/SAS are each their own small section; tools can preserve or omit them independently.

---

### 2.5.1.7 Examples

#### A) Place a DMA buffer in non-cached RAM (no source edits)

- **Mapping path:** RAM Zero (No-Cache)
- **Input pattern:** my_dma.o(.bss.my_buffer)
- **Input section placed in:** .ram_nocache
- **Output section placed in:** __ram_zero_nocache$$

**B) Run a hot ISR from ITCM (copied from FLASH)**

- **Mapping path:** ITCM Code from FLASH
- **Input pattern:** isr_fast.o(.text.my_isr)
- **Input section placed in:** .itcm_code_from_flash
- **Output section placed in:** __itcm_from_flash$$

**C) Put firmware asset in OSPI1 CS1 ReadOnly**

- **Mapping path:** OSPI_CS1 Constant Data
- **Input pattern:** assets.o(.rodata*)
- **Input section placed in:** .ospi1_cs1
- **Output section placed in:** __ospi1_cs1_readonly$$

---

## 2.5.2 FAQ

**Q: Can I still edit the scripts directly?** A: You can, but we **don't recommend it**. Copy to new filenames and INCLUDE them, or maintain a **patch** step. Otherwise, tool-driven mapping/partitioning will be out of sync.

**Q: Will old BSP_PLACE_IN_SECTION() break?** A: It's still supported. A few legacy section names changed; prefer the mapping dialog or update the section names as in the quick translation above.

**Q: How do I share memory between bootloader and app?** A: Use **Solution Partitioning**. The generator will rebuild memory_regions.* and all projects will align automatically.

**Q: What about data that must *not* be re-initialized?** A: Use .noinit/.ram_noinit (or the mapping to **RAM Non-Initialized** → __ram_noinit$$).

**Q: What is the bsp_linker_info.h file located in the <configuration> folder?** A: To support the seamless support of the new linker process, we must pass information to the source code (such as the source and destination of memory initializers). Unfortunately, the current DDSC tooling has limitations that only allow us to generate a header file to this location. Future releases will likely change the location of this information.

---

## 2.5.3 Summary

The new generator brings **consistency, scalability, and zero-touch placement** across toolchains and devices. Start with **Solution Partitioning** and the **Linker Section Mapping** dialog; reserve script edits for exceptional cases via patches or forked includes.

---

## 2.5.4 Appendix: Linker Input Sections

This appendix documents the valid **input** linker_section values that appear in the linker script and may be referenced from user code or configuration macros.

Each section name is based on a **RAM or ROM class**, using a structured naming convention depending on the content type.

**RAM Target Capability Matrix**

| RAM Class | ROM Init | | Zeroed | | NoInit | |
|---|---|---|---|---|---|---|

| | Code | Data | | NoCache | | NoCache |
|---|---|---|---|---|---|---|
| RAM | □ | □ | □ | □ | □ | □ |
| TCM | □ | □ | □ | | □ | |

**ROM Target Capability Matrix**

| ROM Class | Code | ReadOnly | NoInit |
|---|---|---|---|
| ROM | □ | □ | □ |
| FLASH (Gap) | □ | □ | |

### 2.5.4.1 Memory Class Dictionaries

These are the valid memory class identifiers used in linker_section names.

```
ram_class ::= "ram" | "itcm" | "dtcm" | "stcm" | "ctcm" | "sdram" | "ospi0_cs0"
rom_class ::= "flash" | "data_flash" | "qspi_flash" | "ospi0_cs1" | "ospi1_cs1"
```

**The available memory classes depend on the capabilities of your specific device. Not all devices support all RAM or ROM types.**

### 2.5.4.2 RAM-Resident Sections

**Zero-Initialized RAM**

```
.<ram_class>
```

Used for memory cleared to zero at startup.

Examples:

- *(.ram)
- *(.sdram)
- *(.dtcm)

**Uninitialized RAM**

```
.<ram_class>_noinit
.<ram_class>_noinit_nocache
```

Used for reserved memory (e.g., heap/stack) that doesn't require initialization at startup. The _nocache variant is only available on devices with caching hardware.

Examples:

- *(.ram_noinit)
- *(.sdram_noinit_nocache)

**RAM Initialized from ROM**

```
.<ram_class>_code_from_<rom_class>
.<ram_class>_from_<rom_class>
```

Used for RAM-resident code or data loaded from ROM.

Examples:

- *(.ram_code_from_flash)
- *(.dtcm_from_data_flash)
- *(.tcm_code_from_qspi_flash)

### 2.5.4.3 ROM-Resident Sections

**Standard ROM Sections**

```
.<rom_class>
.<rom_class>_code
.<rom_class>_noinit
```

- .rom_class_code: Executable code
- .rom_class: Constants or read-only data
- .rom_class_noinit: Reserved (e.g., alignment)

Examples:

- *(.flash_code)
- *(.data_flash)
- *(.qspi_flash_noinit)

**Flash Gap Sections**

```
.flash_gap_code
.flash_gap_readonly
```

Used only on devices with restricted flash regions (e.g., 0x400–0x43F). These sections support only code and constant data.

### 2.5.4.4 Summary

- linker_section names follow consistent, structured patterns.
- RAM and ROM classes reflect physical memory categories and device-specific capabilities.

- Use _nocache suffixes only for memory that must not be cached (e.g., DMA regions).
- Flash gap sections are a special case for devices with non-contiguous flash address maps.

# Chapter 3 Starting Development

## 3.1 Starting Development Introduction

The wealth of resources available to learn about and use e² studio and FSP can be overwhelming on first inspection, so this section provides a Starting Development Guide with a list of the most important initial steps. Following these highly recommended first 11 steps will bring you up to speed on the development environment in record time. Even experienced developers can benefit from the use of this guide, to learn the terminology that might be unfamiliar or different from previous environments.

1. Read the section What is e² studio?, up to but not including e² studio Prerequisites. This will provide a description of the various windows and views to use e² studio to create a project, add modules and threads, configure module properties, add code, and debug a project. It also describes how to use key coding 'accelerators' like Developer Assist (to drag and drop parameter populated API function calls right into your code), a context aware Autocomplete (to easily find and select from suggested enumerations, functions, types, and many other coding elements), and many other similar productivity enhancers.
2. Read the FSP Architecture, FSP Modules and FSP Stacks sections. These provide the basic background on how FSP modules and stacks are used to construct your application. Understanding their definitions and the theory behind how they combine will make it easier to develop with FSP.
3. Read a few Modules sections to see how to use API function calls, structures, enumerations, types and callbacks. These module guides provide the information you will use to implement your project code.
4. After you have a Kit and you have downloaded and installed e² studio and FSP, you can build and debug a simple project to test your installation, tool flow, and the kit. (If you do not have a Kit or have not yet installed the development software, use the links included in the e² studio Prerequisites for more information.) The simple Tutorial: Your First RA MCU Project - Blinky will Blink an LED on and off. Follow the instructions for importing and running this project in section Create a New Project for Blinky. It will use some of the key steps for managing projects within e² studio and is a good way to learn the basics. If you are using a multicore RA MCU then you will want to read Tutorial: Your First RA Multicore Application - Blinky for information on creating a multicore blinky application.
5. Once you have successfully run Blinky you have a good starting point for using FSP for more complex projects. The Using HAL Drivers Tutorial, available at Tutorial: Using HAL Drivers - Programming the WDT, shows how to create a project from scratch, using FSP API functions. Do this next.
6. Several Hands-on Quick FSP Labs are available that cover key development topics with short 15-minute Do it Yourself (DiY) activities targeting the EK-RA6M3. Topics covered include code development accelerators like Developer Assistance, Autocomplete, Help, Visual Expressions and using Example Projects. The complete list of available Quick FSP Labs can be found here: https://en-support.renesas.com/knowledgeBase/category/31087/subcategory/31090. Doing a couple of these labs provides further details on using FSP, and is also good practice. Running these labs is highly recommended.
7. The balance of the FSP Architecture sections (that is, those not called out in step 2 above) contain additional reference material that may be helpful in the future. Scan them so you know what they contain, in case you need them.

8. The balance of the e² studio User Guide, starting with the What is a Project? section up to, but not including, Writing the Application section, provides a detailed description of each of the key steps, windows, and entries used to create, manage, configure, build and debug a project. Much of this may be familiar after running through the tutorials and Quick Labs. However, it is important to have a good grasp of what each of the configuration tabs are used for as that is where the bulk of the project preparation work takes place prior to writing code. Skim over this section as it may help with any questions in the future.
9. Read the Writing the Application section to get a short introduction to the steps used when creating application code with FSP. It covers both RTOS-independent and RTOS-dependent applications. It also includes a short description for several of the code accelerators you should be familiar with by now. Using additional Quick FSP Labs is a good way to become familiar with the application development process and links to them are included in the appropriate places in this section. You can find the complete list of available Quick FSP Labs here: https://en-support.renesas.com/knowledgeBase/19308277.
10. Scan the Debugging the Project section to see the steps required to download and start a debug session.
11. Explore the additional material available on the following web pages and bookmark the resources that look most valuable to you:
    a. RA Landing Page: https://www.renesas.com/ra
    b. FSP Landing Page: https://www.renesas.com/fsp
    c. Example Projects on GitHub: https://github.com/renesas/ra-fsp-examples
    d. Quick FSP Labs Listing: https://en-support.renesas.com/knowledgeBase/19308277
    e. RA and FSP Knowledge Base (with articles of interest on RA and FSP): https://en-support.renesas.com/knowledgeBase/category/31087
    f. RA and FSP Renesas Rulz site (Community posted and answered questions): https://community.renesas.com/mcu-mpu/ra/
    g. FSP Releases: https://github.com/renesas/fsp/releases
    h. FSP Documentation: https://renesas.github.io/fsp
    i. Online Technical Support: https://www.renesas.com/us/en/support/contact.html

# 3.2 e² studio User Guide

## 3.2.1 What is e² studio?

Renesas e² studio is a development tool encompassing code development, build, and debug. e² studio is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT).

When developing for RA MCUs, e² studio hosts the Renesas Flexible Software Package (FSP). FSP provides a wide range of time saving tools to simplify the selection, configuration, and management of modules and threads, to easily implement complex applications. The time saving tools available in e² studio and FSP include the following:

- A Graphical User Interface (GUI) (see Adding Threads and Drivers) with numerous wizards for configuring and auto-generating code
- A context sensitive Autocomplete (see Tutorial: Using HAL Drivers - Programming the WDT) feature that provides intelligent options for completing a programming element
- A Developer Assistance tool for selection of and drag and drop placement of API functions directly in application code
- A Welcome Window with links to example projects, application notes and a variety of other self-help support resources
- An Information Icon from each module is provided in the graphic configuration viewer that links to specific design resources, including code 'cheat sheets' that provide useful starting points for common application implementations.

Figure 6: e² studio Splash Screen

e² studio organizes project work based on Perspectives, Views, Windows, Panes, and Pages (sometimes called Tabs). A window is a section of the e² studio GUI that presents information on a key topic. Windows often use tabs to select sub-topics. For example, an editor window might have a tab available for each open file, so it is easy to switch back and forth between them. A window Pane is a section of a window. Within a window, multiple Panes can be opened and viewed simultaneously, as opposed to a tabbed window, where only individual content is displayed. A memory-display Window, for example, might have multiple Panes that allow the data to be displayed in different formats, simultaneously. A Perspective is a collection of Views and Windows typical for a specific stage of development. The default perspectives are a C/C++ Perspective, an FSP Configuration Perspective and a Debug Perspective. These provide specific Views, Windows, Tabs, and Panes tailored for the common tasks needed during the specific development stage.

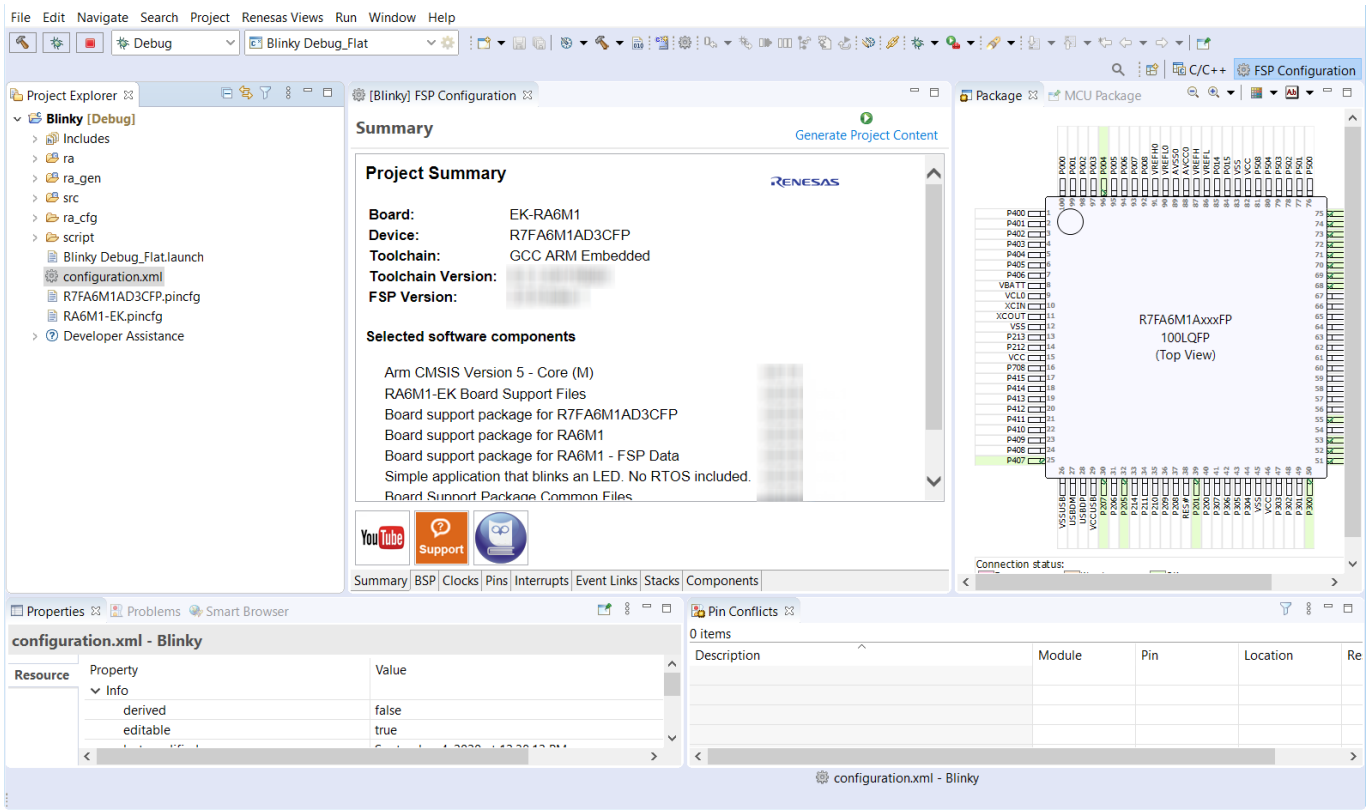



Figure 7: Default Perspective

In addition to managing project development, selecting modules, configuring them and simplifying code development, e² studio also hosts the engine for automatically generating code based on module selections and configurations. The engine continually checks for dependencies and automatically adds any needed lower level modules to the module stack. It also identifies any lower level modules that require configuration (for example, an interrupt that needs to have a priority assigned). It also provides a guide for selecting between multiple choices or options to make it easy to complete a fully functional module stack.

The Generate Project Content function takes the selected and configured modules and automatically generates the complete and correct configuration code. The code is added to the folders visible in the **Project Explorer** window in e² studio. The configuration.xml file in the project folder holds all the generated configuration settings. This file can be opened in the GUI-based RA Configuration editor to make further edits and changes. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required using this editor.

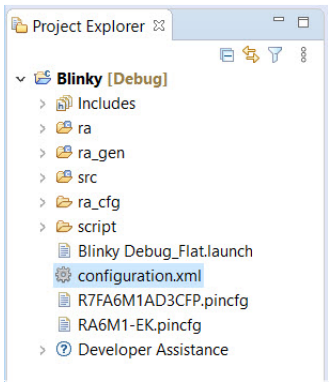



Figure 8: Project Explorer Window showing generated folders and configuration.xml file

## 3.2.2 e² studio Prerequisites

### 3.2.2.1 Obtaining an RA MCU Kit

To develop applications with FSP, start with one of the Renesas RA MCU Evaluation Kits. The Renesas RA MCU Evaluation Kits are designed to seamlessly integrate with e² studio.

Ordering information, Quick Start Guides, User Manuals, and other related documents for all RA MCU Evaluation Kits are available at https://www.renesas.com/ra.

### 3.2.2.2 PC Requirements

The following are the minimum PC requirements to use e² studio:

- Windows 10 with Intel i5 or i7, or AMD A10-7850K or FX
- Memory: 8-GB DDR3 or DDR4 DRAM (16-GB DDR4/2400-MHz RAM is preferred)
- Minimum 250-GB hard disk

### 3.2.2.3 Installing e² studio, platform installer and the FSP package

Detailed installation instructions for e² studio and FSP are available on the Renesas website https://www.renesas.com/fsp. Review the release notes for e² studio to ensure that the e² studio version supports the selected FSP version. The starting version of the installer includes all features of the RA MCUs.

#### 3.2.2.4 Choosing a Toolchain

e² studio can work with several toolchains and toolchain versions such as the GNU Arm compiler, Arm AC6, and LLVM Embedded Toolchain for Arm. Versions of the GNU Arm and LLVM compilers verified for use with FSP are included in the e² studio installer.

#### 3.2.2.5 Licensing

FSP licensing includes full source code, limited to Renesas hardware only.

### 3.2.3 What is a Project?

In e² studio, all FSP applications are organized in RA MCU projects. Setting up an RA MCU project involves:

1. Creating a Project
2. Configuring a Project

These steps are described in detail in the next two sections. When you have existing projects already, after you launch e² studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** window. Each project has an associated configuration file named configuration.xml, which is located in the project's root directory.

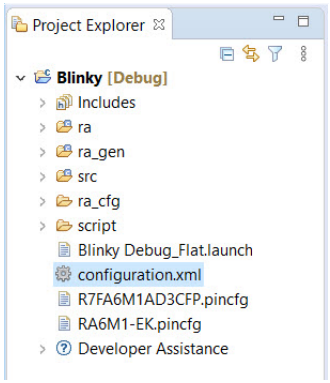



Figure 9: e² studio Project Configuration file

Double-click on the configuration.xml file to open the RA MCU Project Editor. To edit the project configuration, make sure that the **FSP Configuration** perspective is selected in the upper right hand corner of the e² studio window. Once selected, you can use the editor to view or modify the configuration settings associated with this project.

Figure 10: e² studio FSP Configuration Perspective

*Note*

*Whenever the RA project configuration (that is, the configuration.xml file) is saved, a verbose RA Project Report file (ra_cfg.txt) with all the project settings is generated. The format allows differences to be easily viewed using a text comparison tool. The generated file is located in the project root directory.*

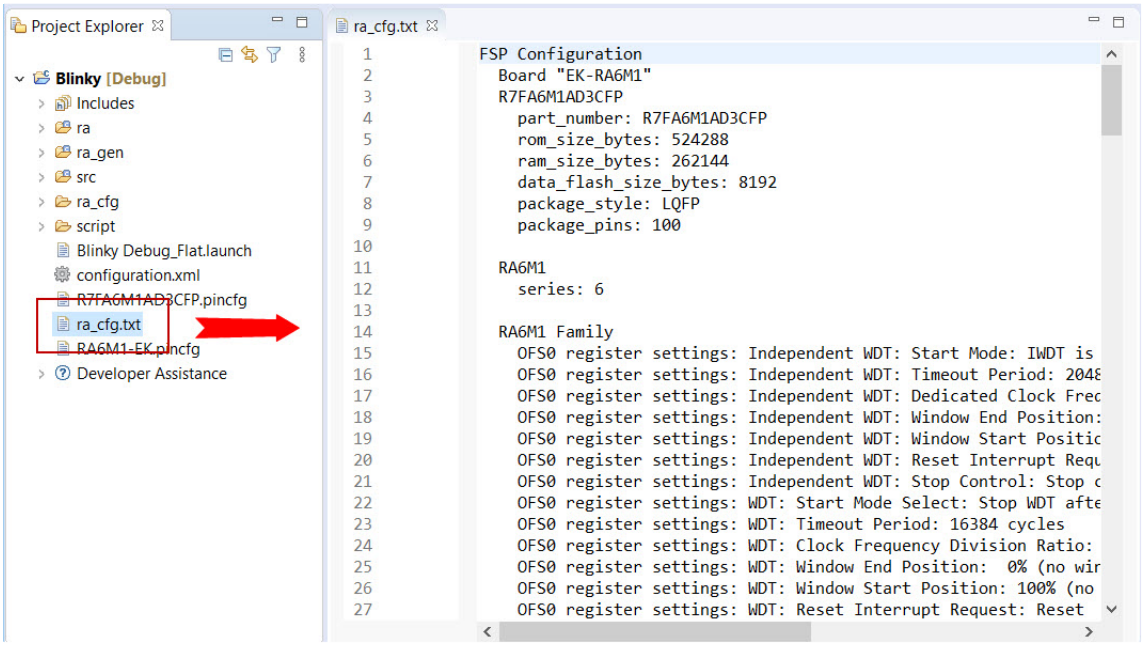



Figure 11: RA Project Report

The RA Project Editor has a number of tabs. The configuration steps and options for individual tabs are discussed in the following sections.

*Note*

*The tabs available in the RA Project Editor depend on the e² studio version and the layout may vary slightly, however the functionality should be easy to follow..*

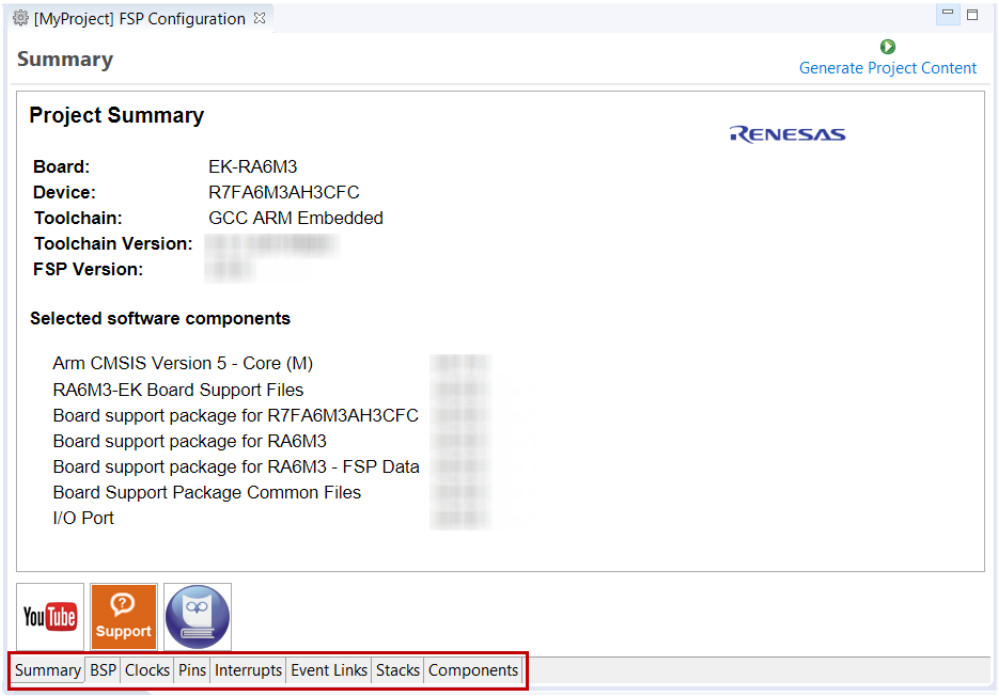



Figure 12: RA Project Editor tabs

- Click on the YouTube icon to visit the Renesas FSP playlist on YouTube
- Click on the Support icon to visit RA support pages at Renesas.com
- Click on the user manual (owl) icon to open the RA software package User's Manual

## 3.2.4 Creating a Project

During project creation, you specify the type of project, give it a project name and location, and configure the project settings for version, target board, whether an RTOS is included, the toolchain version, and the beginning template. This section includes easy-to-follow step-by-step instructions for all of the project creation tasks. Once you have created the project, you can move to configuring the project hardware (clocks, pins, interrupts) and the parameters of all the modules that are part of your application.

### 3.2.4.1 Creating a New Project

For RA MCU applications, generate a new project using the following steps:

1. Click on **File > New > RA C/C++ Project > Renesas RA**.

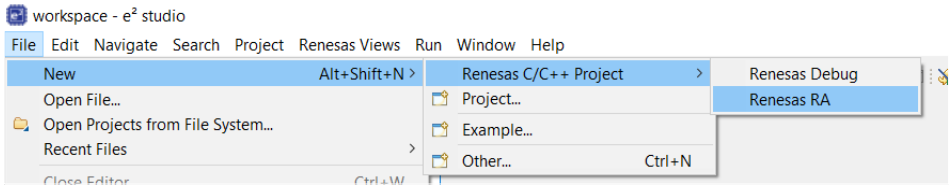



Figure 13: New RA MCU Project

Then click on the type of template for the type of project you are creating.

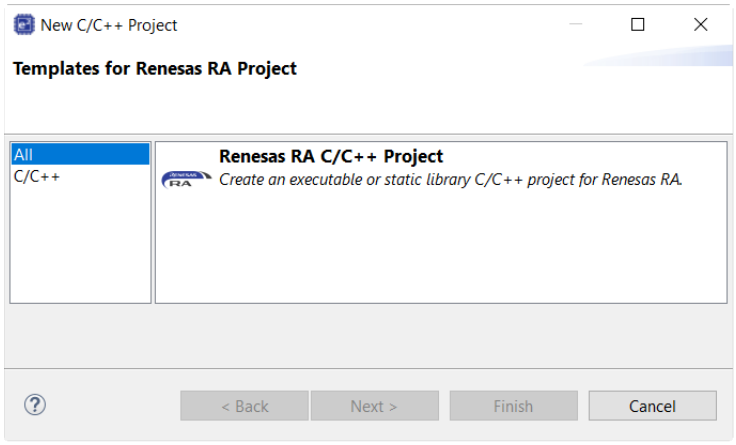



Figure 14: New Project Templates

2. Select a project name and location.

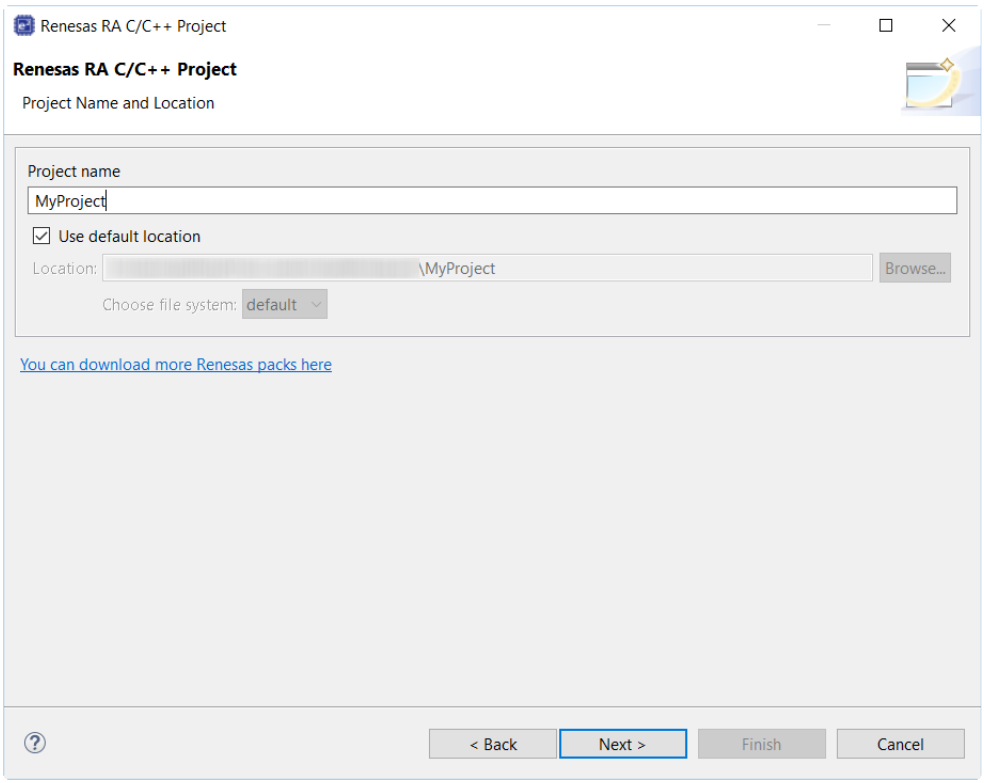



Figure 15: RA MCU Project Generator (Screen 1)

3. Click **Next**.

#### 3.2.4.2 Selecting a Board and Toolchain

In the **Project Configuration** window select the hardware and software environment:

1. Select the **FSP version**.
2. Select the **Board** for your application. You can select an existing RA MCU Evaluation Kit or select **Custom User Board** for any of the RA MCU devices with your own BSP definition.
3. Select the **Device**. The **Device** is automatically populated based on the **Board** selection. Only change the **Device** when using the **Custom User Board (Any Device)** board selection.
4. To add threads, select **RTOS**, or **No RTOS** if an RTOS is not being used.
5. The **Toolchain** selection defaults to **GCC Arm Embedded**.
6. Select the **Toolchain version**. This should default to the installed toolchain version.
7. Select the **Debugger**. The J-Link Arm Debugger is preselected.

8. Click **Next**.

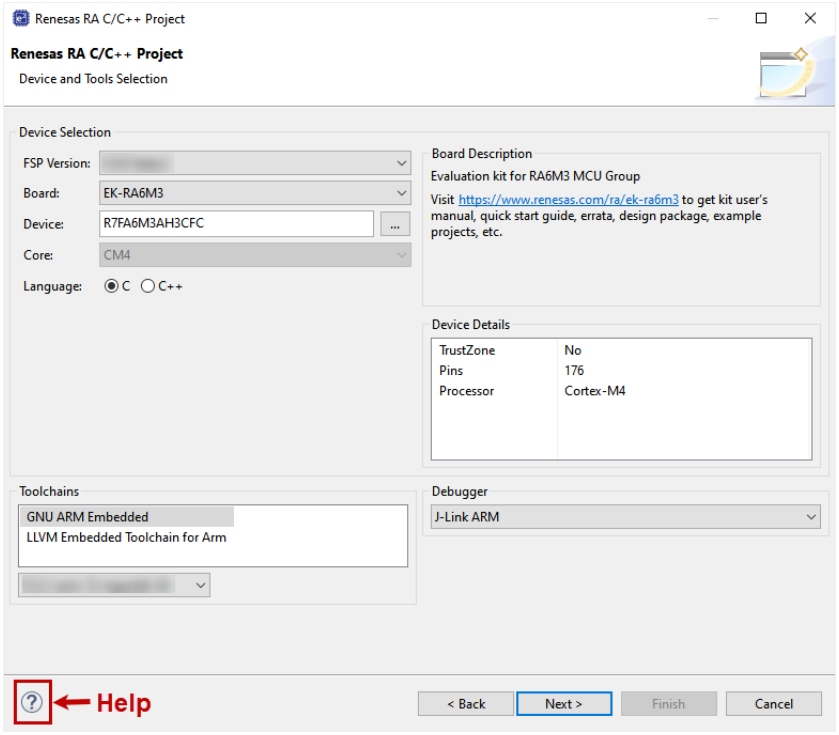



Figure 16: RA MCU Project Generator (Screen 2)

*Note*

*Click on the **Help** icon (?) for user guides, RA contents, and other documents.*

### 3.2.4.3 Selecting Flat or Arm TrustZone Project

If you selected a device or tool based on an Arm® Cortex®-M33, you next select whether to use Arm® TrustZone® technology in your project. For normal, non-TrustZone projects, select "Flat".

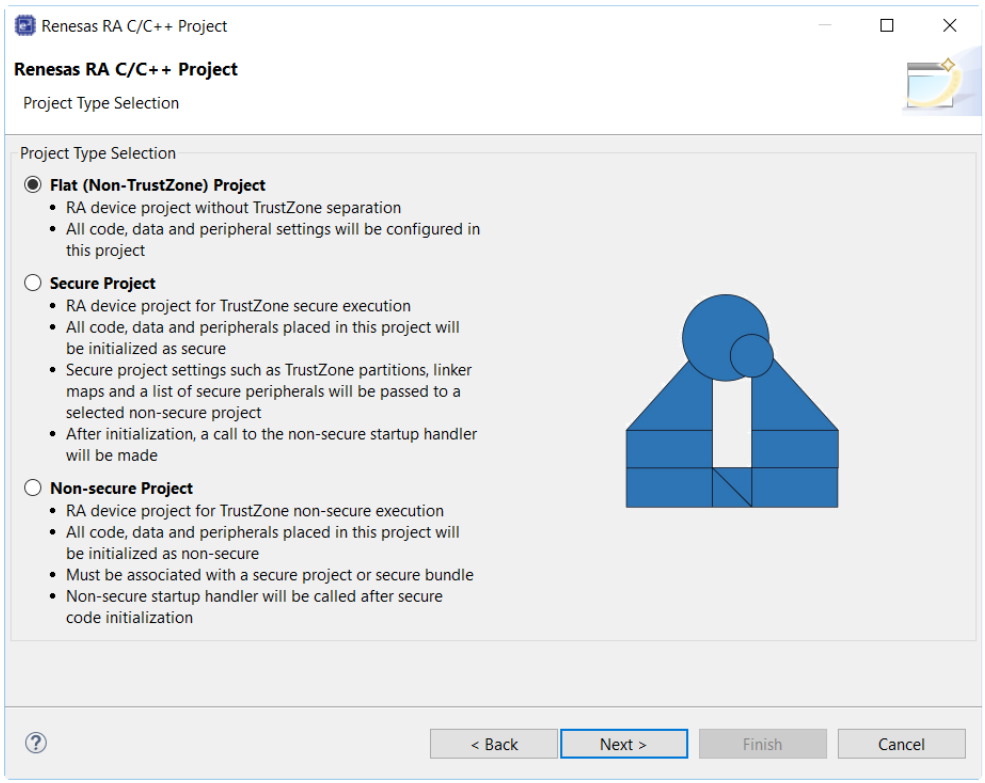



Figure 17: Flat, Secure, or Non-Secure Project

For more information on Arm TrustZone technology, see Primer: Arm TrustZone Project Development.

### 3.2.4.4 Selecting a Project Template

In the next window, select the buiild artifact and RTOS.

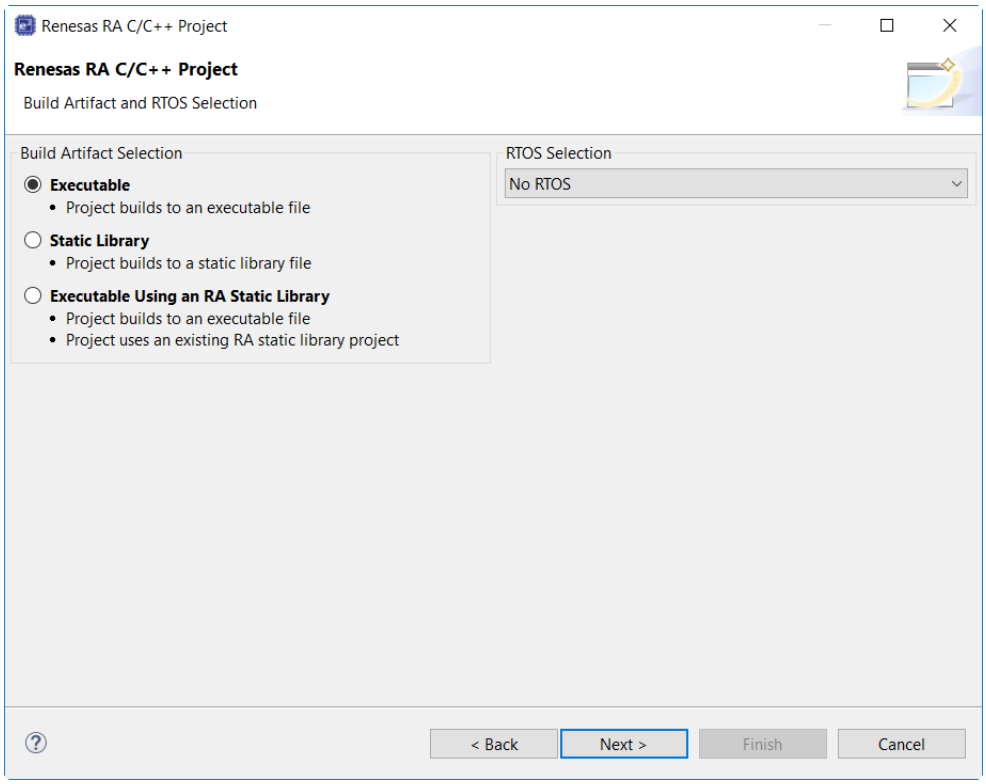



Figure 18: RA MCU Project Generator (Screen 3)

In the next window, select a project template from the list of available templates. By default, this screen shows the templates that are included in your current RA MCU pack. Once you have selected the appropriate template, click **Finish**.

*Note*

*If you want to develop your own application, select the basic template for your board,* ***Bare Metal - Minimal****.*

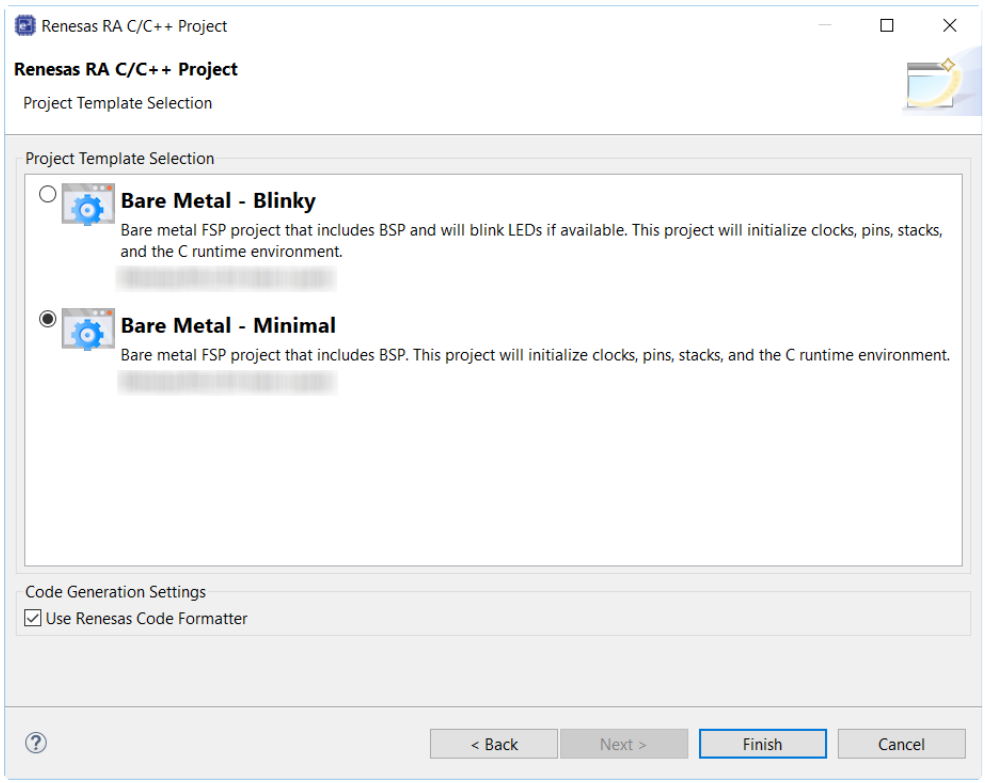



Figure 19: RA MCU Project Generator (Screen 4)

When the project is created, e² studio displays a summary of the current project configuration in the RA MCU Project Editor.

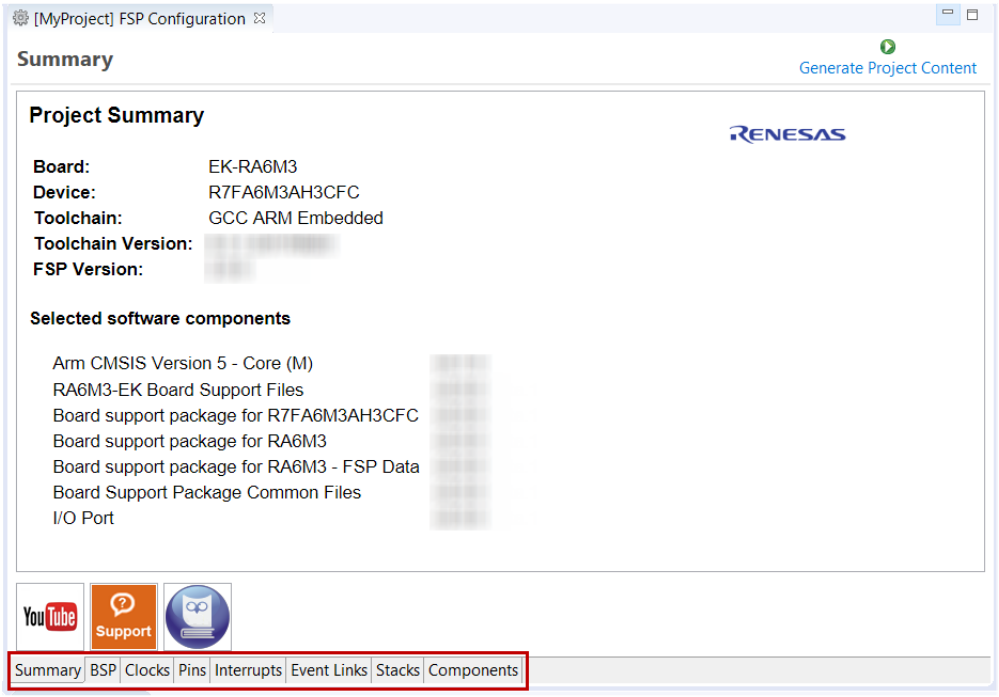



Figure 20: RA MCU Project Editor and available editor tabs

On the bottom of the RA MCU Project Editor view, you can find the tabs for configuring multiple aspects of your project:

- With the **Summary** tab, you can see all they key characteristics of the project: board, device, toolchain, and more.
- With the **BSP** tab, you can change board specific parameters from the initial project selection.
- With the **Clocks** tab, you can configure the MCU clock settings for your project.
- With the **Pins** tab, you can configure the electrical characteristics and functions of each port pin.
- With the **Interrupts** tab, you can add new user events/interrupts.
- With the **Event Links** tab, you can configure events used by the Event Link Controller.
- With the **Stacks** tab, you can add and configure FSP modules. For each module selected in this tab, the **Properties** window provides access to the configuration parameters, interrupt priorities, and pin selections.
- The **Components** tab provides an overview of the selected modules. Although you can also add drivers for specific FSP releases and application sample code here, this tab is normally only used for reference.

The functions and use of each of these tabs is explained in detail in the next section.

## 3.2.5 Configuring a Project

Each of the configurable elements in an FSP project can be edited using the appropriate tab in the RA Configuration editor window. Importantly, the initial configuration of the MCU after reset and before any user code is executed is set by the configuration settings in the **BSP**, **Clocks** and **Pins** tabs. When you select a project template during project creation, e² studio configures default values that are appropriate for the associated board. You can change those default values as needed. The following sections detail the process of configuring each of the project elements for each of the associated tabs.

### 3.2.5.1 Summary Tab

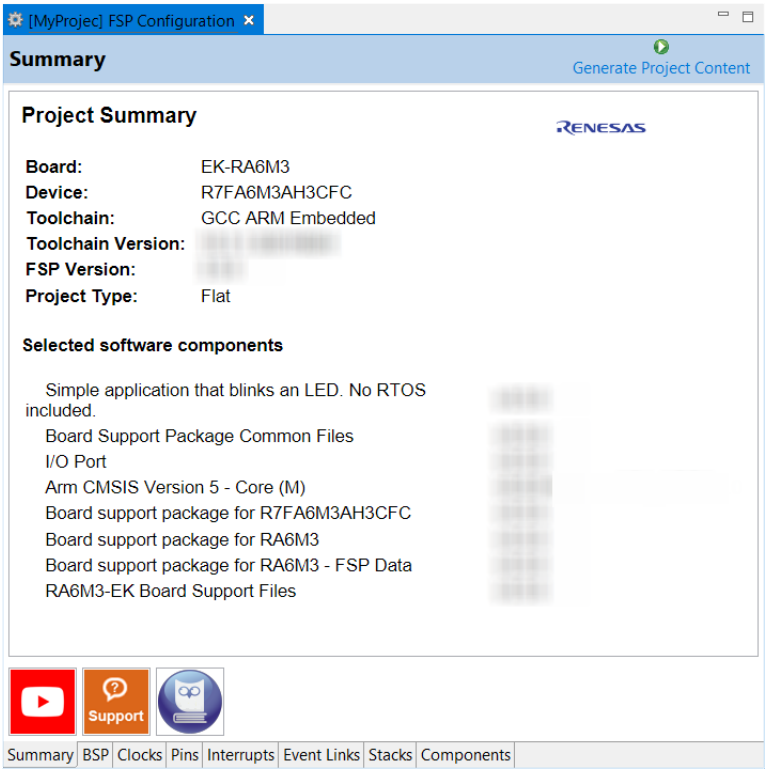



Figure 21: Configuration Summary tab

The **Summary** tab, seen in the above figure, identifies all the key elements and components of a project. It shows the target board, the device, toolchain and FSP version. Additionally, it provides a list of all the selected software components and modules used by the project. This is a more convenient summary view when compared to the **Components** tab.

The summary tab also includes handy icons with links to the Renesas YouTube channel, the Renesas support page and to the RA FSP User Manual that was downloaded during the installation process.

### 3.2.5.2 Configuring the BSP

The **BSP** tab shows the currently selected board (if any) and device. The Properties view is located in the lower left of the Project Configurations view as shown below.

*Note*

*If the Properties view is not visible, click **Window** > **Show View** > **Properties** in the top menu bar.*

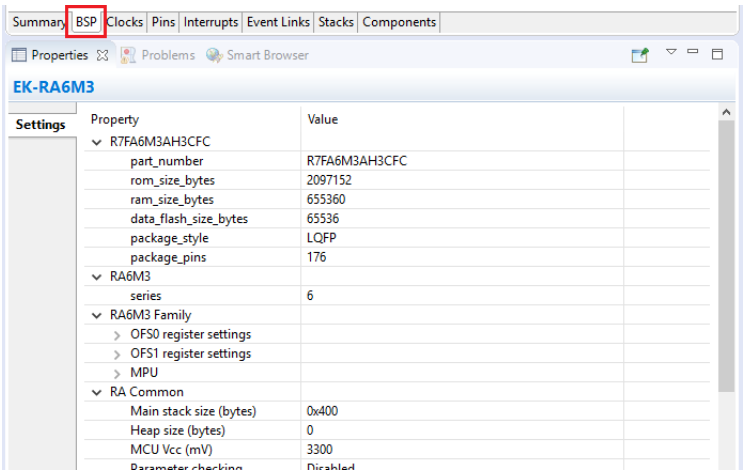



Figure 22: Configuration BSP tab

The **Properties** view shows the configurable options available for the BSP. These can be changed as required. The BSP is the FSP layer above the MCU hardware. e² studio checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you click the **Generate Project Content** button, the BSP configuration contents are written to ra_cfg/fsp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

### 3.2.5.3 Configuring Clocks

The **Clocks** tab presents a graphical view of the MCU's clock tree, allowing the various clock dividers and sources to be modified. If a clock setting is invalid, the offending clock value is highlighted in red. It is still possible to generate code with this setting, but correct operation cannot be guaranteed. In the figure below, the USB clock UCLK has been changed so the resulting clock frequency is 60 MHz instead of the required 48 MHz. This parameter is colored red.

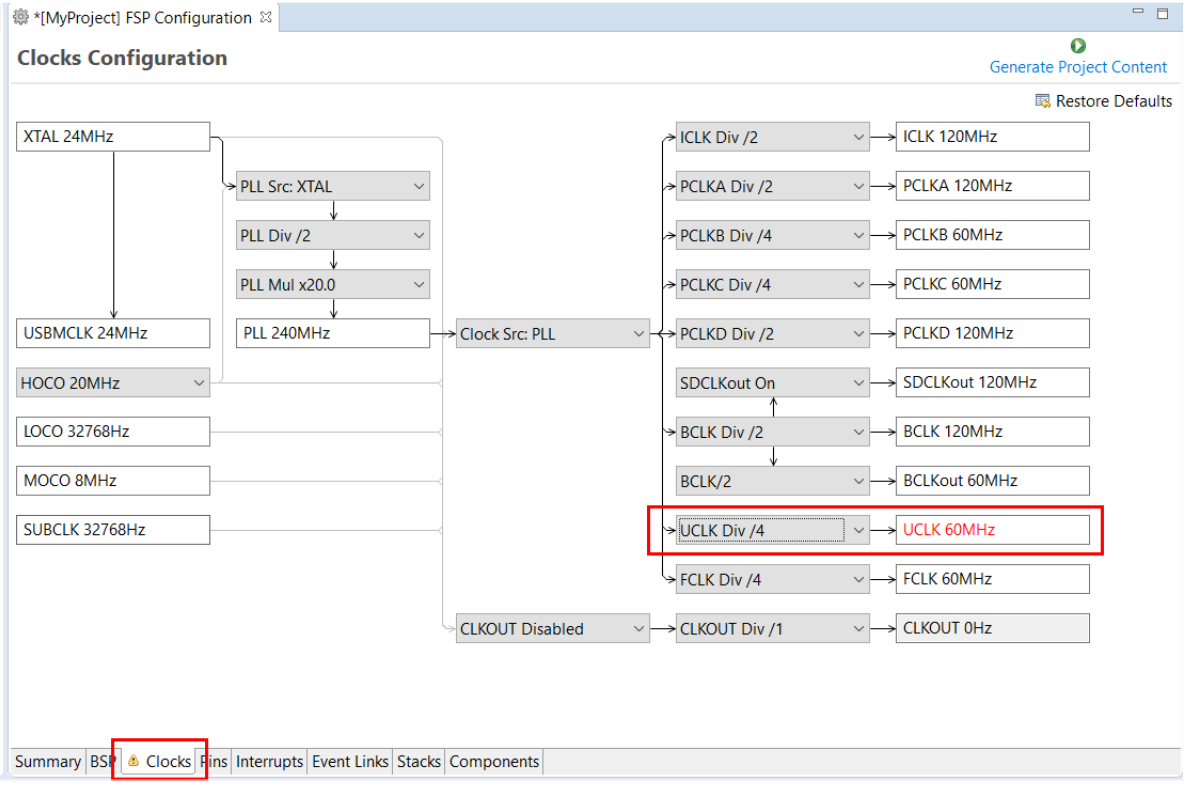



Figure 23: Configuration Clocks tab

When you click the **Generate Project Content** button, the clock configuration contents are written to: ra_gen/bsp_clock_cfg.h

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

### 3.2.5.4 Configuring Pins

The **Pins** tab provides flexible configuration of the MCU's pins. As many pins are able to provide multiple functions, they can be configured on a peripheral basis. For example, selecting a serial channel via the SCI peripheral offers multiple options for the location of the receive and transmit pins for that module and channel. Once a pin is configured, it is shown as green in the **Package** view.

*Note*

*If the **Package** view window is not open in e² studio, select **Window** > **Show View** > **Pin Configurator** > **Package** from the top menu bar to open it.*

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as the EK-RA6M3, some peripherals connected on the board are preselected.

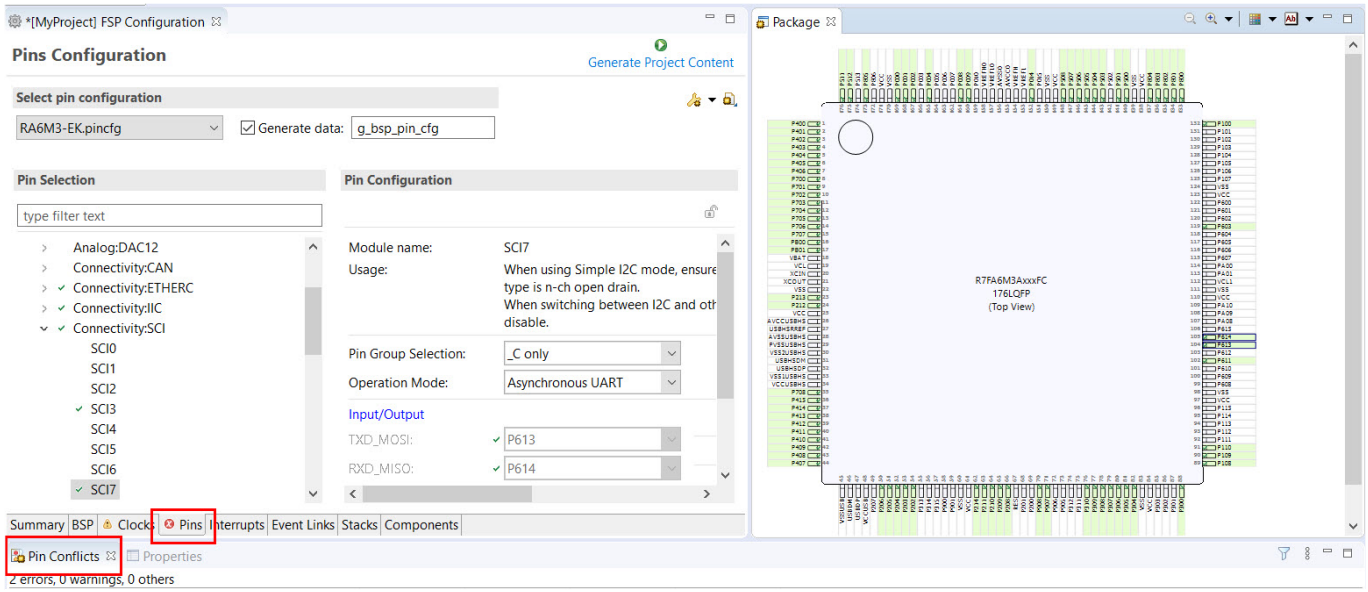



Figure 24: Pins Configuration

The pin configurator includes a built-in conflict checker, so if the same pin is allocated to another peripheral or I/O function the pin will be shown as red in the package view and also with white cross in a red square in the **Pin Selection** pane and **Pin Configuration** pane in the main **Pins** tab. The **Pin Conflicts** view provides a list of conflicts, so conflicts can be quickly identified and fixed.

In the example shown below, port P611 is already used by the CAC, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the CAC in the **Pin Selection** pane on the left side of the tab.

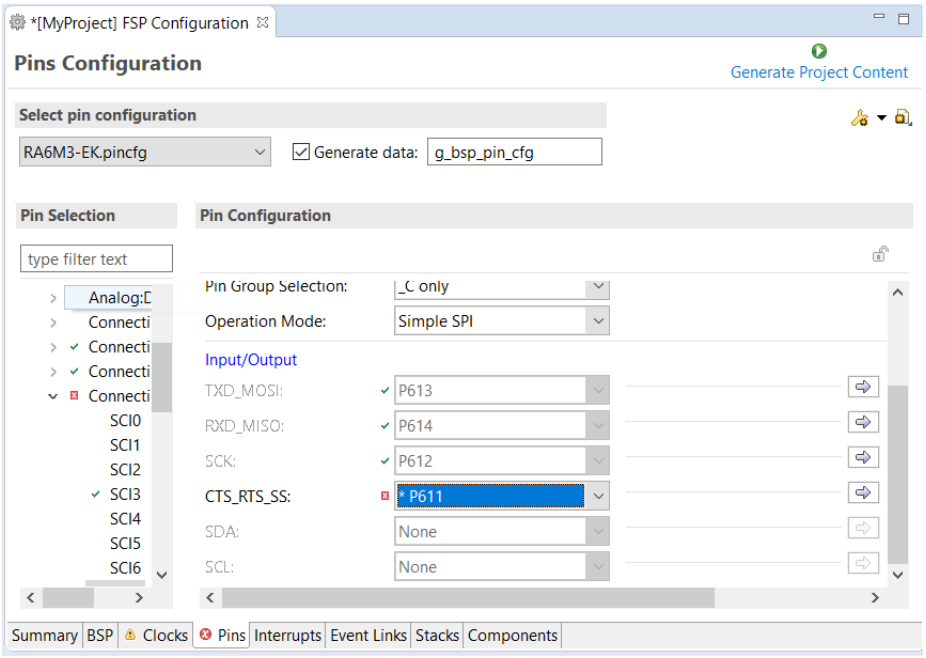



Figure 25: e² studio Pin configurator

The pin configurator also shows a package view and the selected electrical or functional characteristics of each pin.

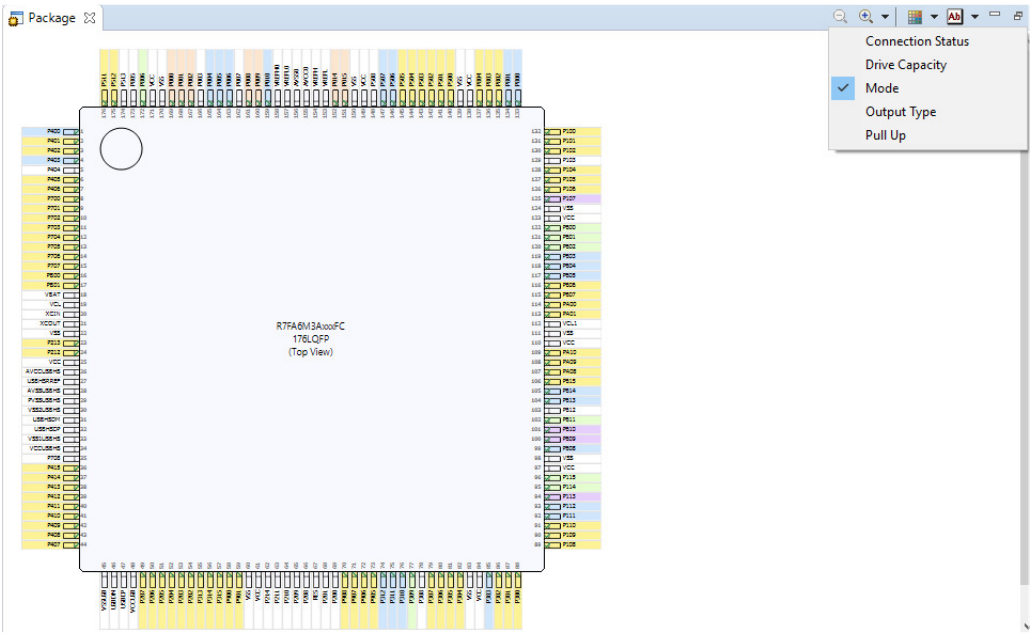



Figure 26: e² studio Pin configurator package view

When you click the **Generate Project Content** button, the pin configuration contents are written to: ra_gen\bsp_pin_cfg.h

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

To make it easy to share pinning information for your project, e² studio exports your pin configuration settings to a csv format and copies the csv file to ra_gen/<MCU package>.csv.

### 3.2.5.5 Configuring Interrupts from the Stacks Tab

You can use the **Properties** view in the **Stacks** tab to enable interrupts by setting the interrupt priority. Select the driver in the **Stacks** pane to view and edit its properties.

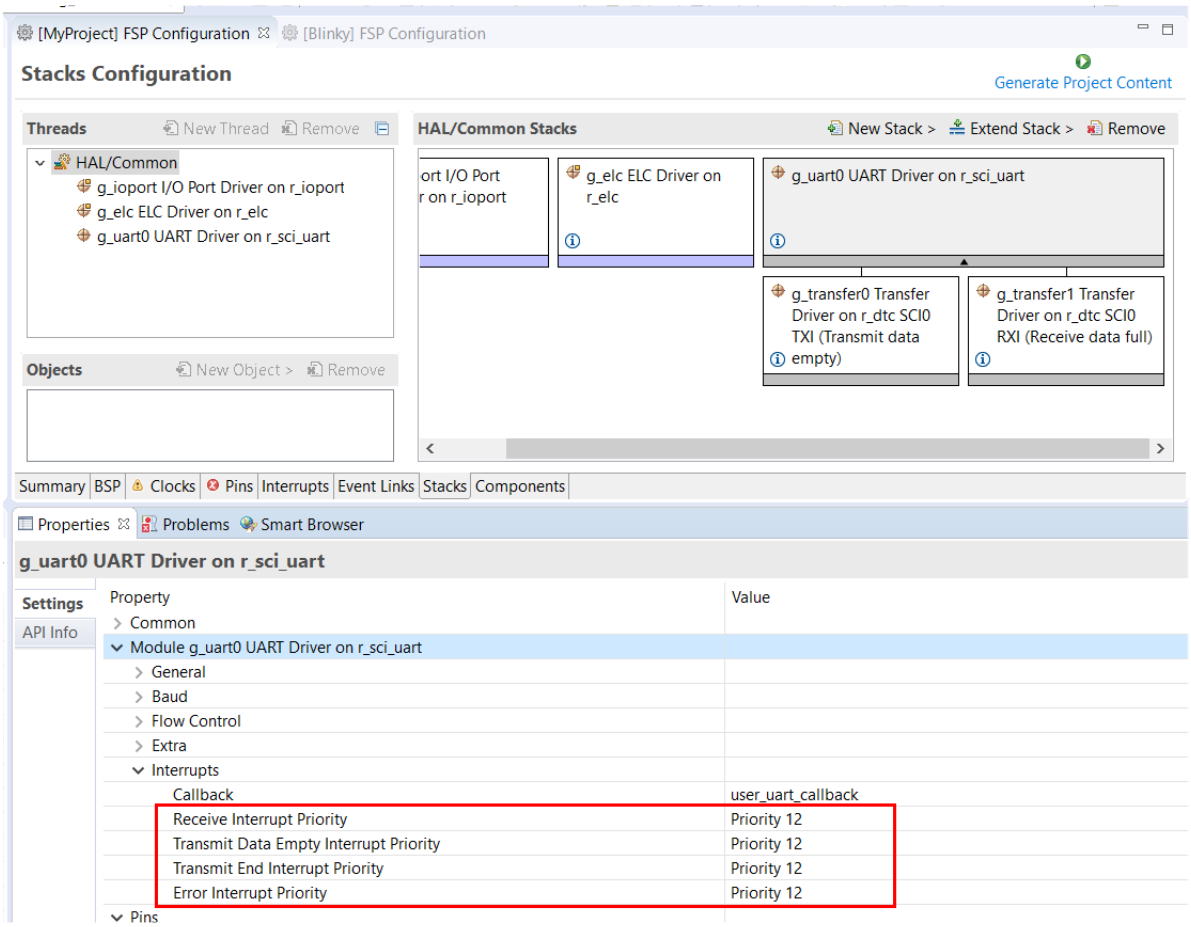



Figure 27: Configuring Interrupts in the Stacks tab

### Creating Interrupts from the Interrupts Tab

On the **Interrupts** tab, the user can bypass a peripheral interrupt set by FSP by setting a user-defined ISR. This can be done by adding a new event via the **New User Event** button.

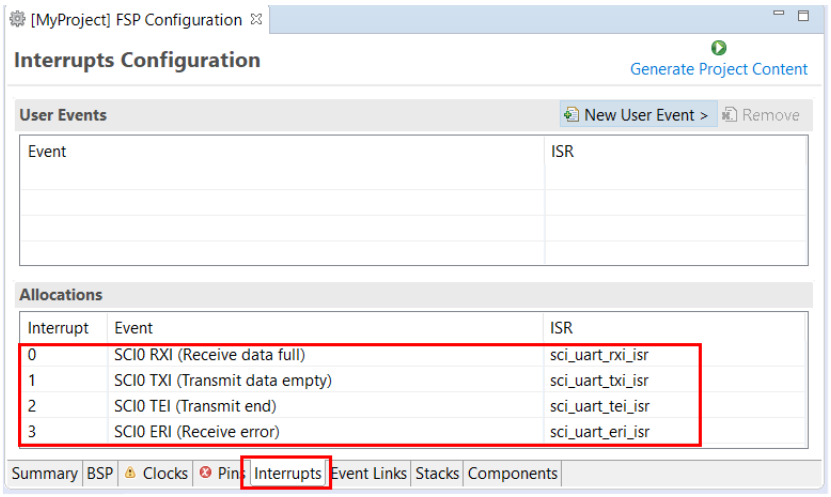



Figure 28: Configuring interrupt in Interrupt Tab

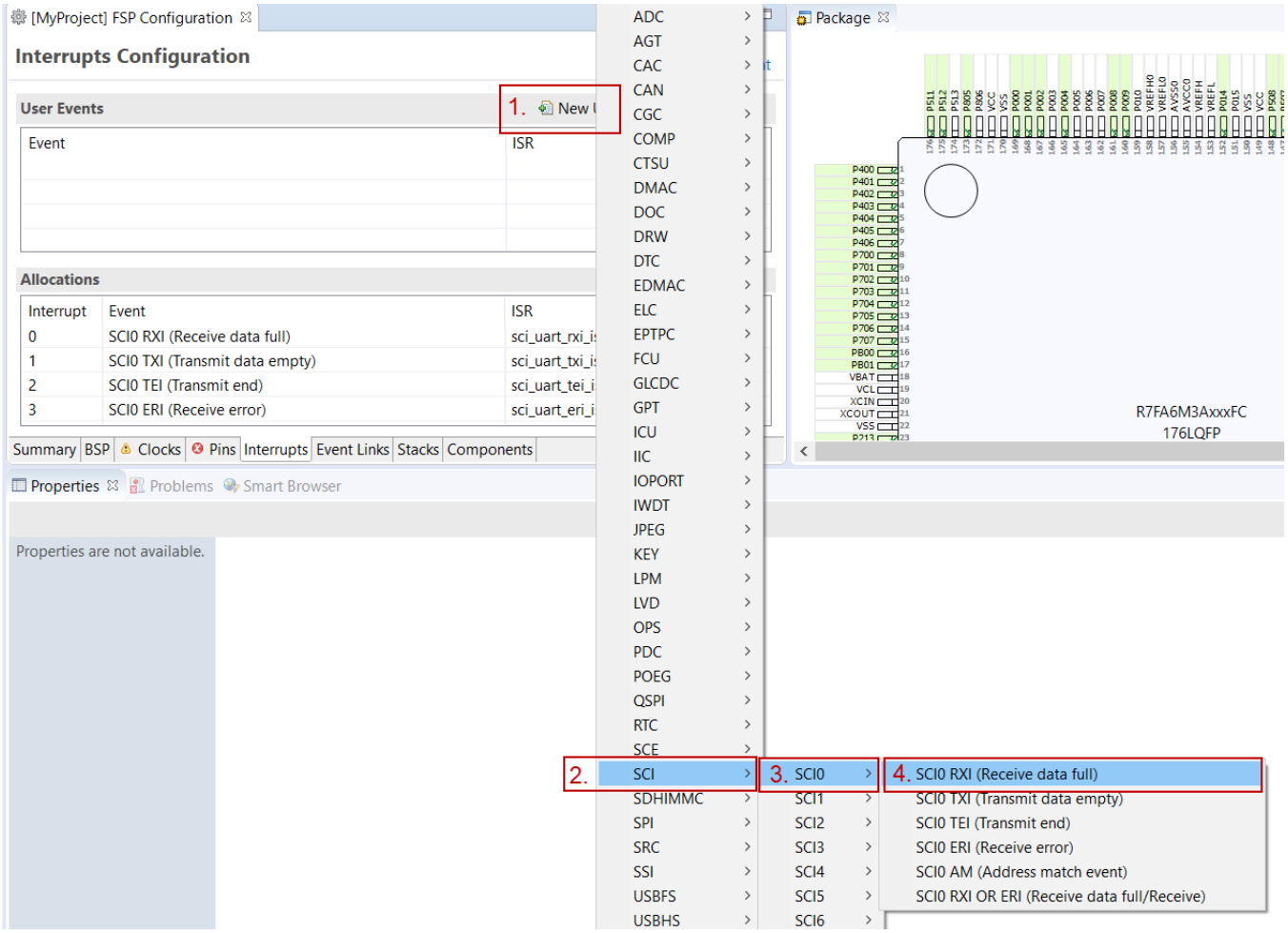



Figure 29: Adding user-defined event

Enter the name of ISR for the new user event.

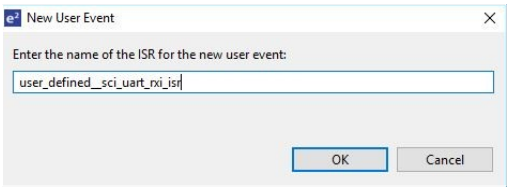



Figure 30: User-defined event ISR

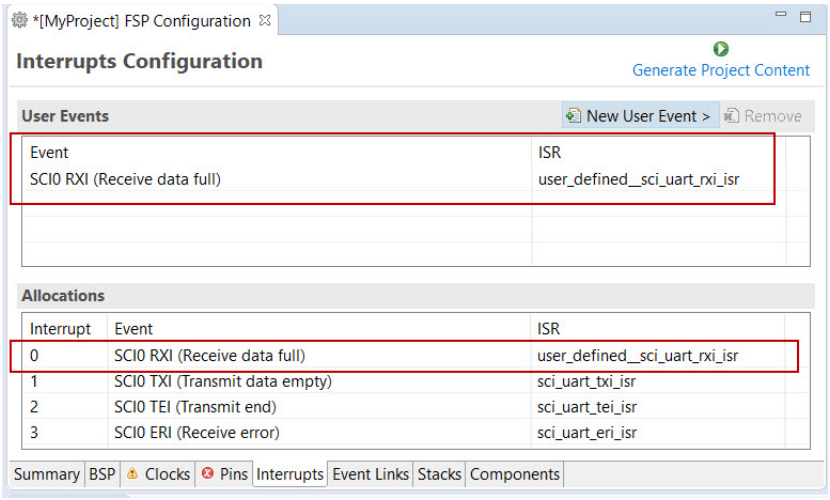



Figure 31: Using a user-defined event

### 3.2.5.6 Viewing Event Links

The Event Links tab can be used to view the Event Link Controller events. The events are sorted by peripheral to make it easy to find and verify them.

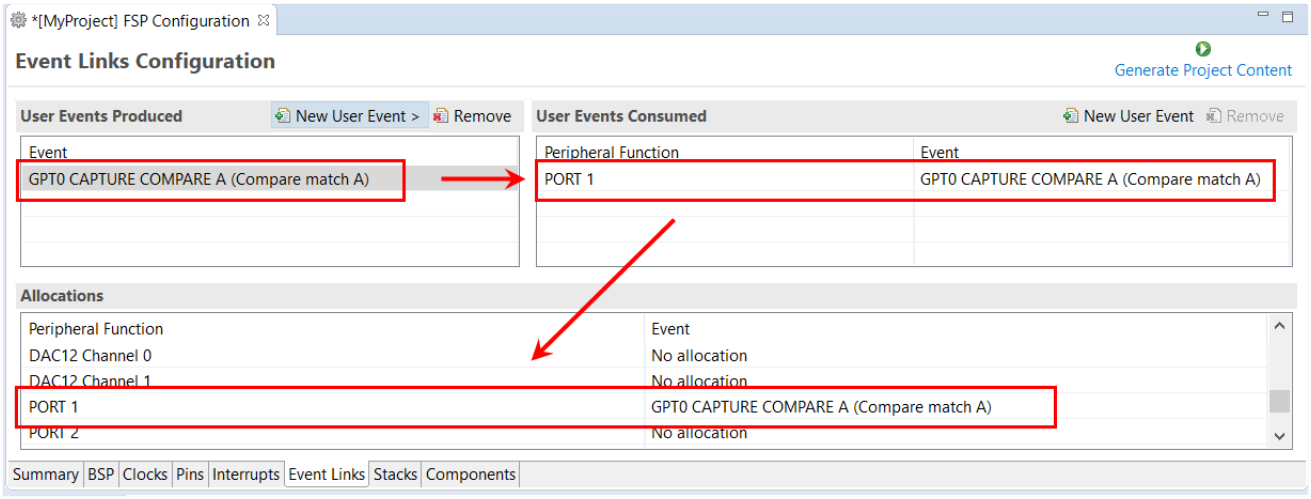



Figure 32: Viewing Event Links

Like the Interrupts tab, user-defined event sources and destinations (producers and consumers) can be defined by clicking the relevant **New User Event** button. Once a consumer is linked to a producer the link will appear in the **Allocations** section at the bottom.

*Note*

*When selecting an ELC event to receive for a module (or when manually defining an event link), only the events that are made available by the modules configured in the project will be shown.*

## 3.2.6 Adding Threads and Drivers

Every RTOS-based RA Project includes at least one RTOS Thread and a stack of FSP modules running

in that thread. The **Stacks** tab is a graphical user interface which helps you to add the right modules to a thread and configure the properties of both the threads and the modules associated with each thread. Once you have configured the thread, e² studio automatically generates the code reflecting your configuration choices.

For any driver, or, more generally, any module that you add to a thread, e² studio automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Stacks pane, which e² studio populates with the selected modules and module options for the selected thread.

The default view of the **Stacks** tab includes a Common Thread called **HAL/Common**. This thread includes the driver for I/O control (IOPORT). The default stack is shown in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common driver are special in that FSP only requires a single instance of each, which e² studio then includes in every user-defined thread by default.

In applications that do not use an RTOS or run outside of the RTOS, the HAL/Common thread becomes the default location where you can add additional drivers to your application.

For a detailed description on how to add and configure modules and stacks, see the following sections:

- Adding and Configuring HAL Drivers
- Adding Drivers to a Thread and Configuring the Drivers

Once you have added a module either to HAL/Common or to a new thread, you can access the driver's configuration options in the **Properties** view. If you added thread objects, you can access the objects configuration options in the **Properties** view in the same way.

You can find details about how to configure threads here: Configuring Threads

*Note*

*Driver and module selections and configuration options are defined in the FSP pack and can therefore change when the FSP version changes.*

### 3.2.6.1 Adding and Configuring HAL Drivers

For applications that run outside or without the RTOS, you can add additional HAL drivers to your application using the HAL/Common thread. To add drivers, follow these steps:

1. Click on the HAL/Common icon in the **Stacks** pane. The Modules pane changes to **HAL/Common Stacks**.

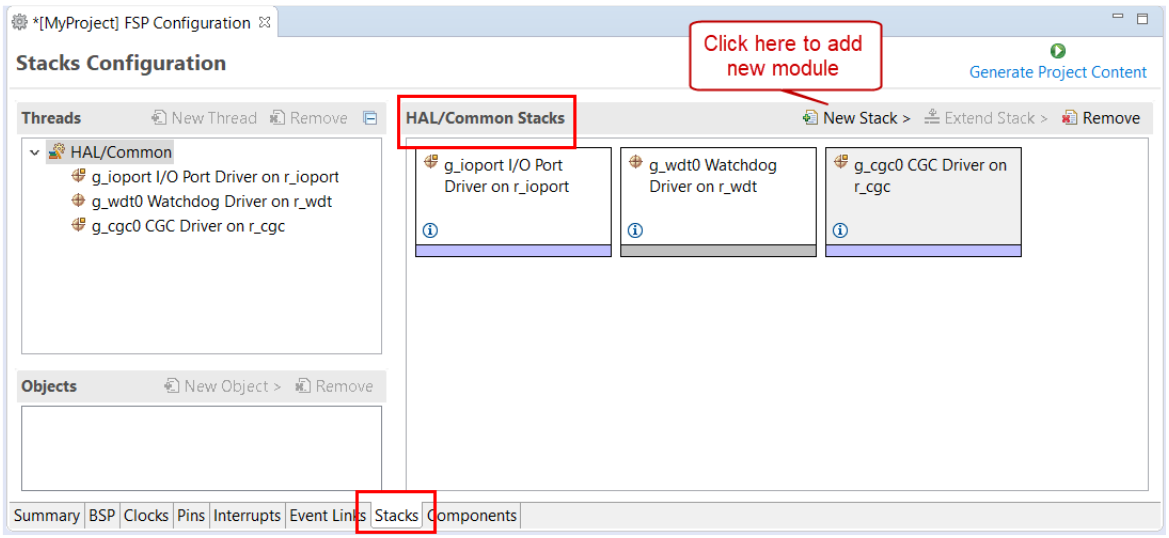



Figure 33: e² studio Project configurator - Adding drivers

2. Click **New Stack** to see a drop-down list of HAL level drivers available in FSP.

3. Select a driver from the menu **New Stack > Driver**.

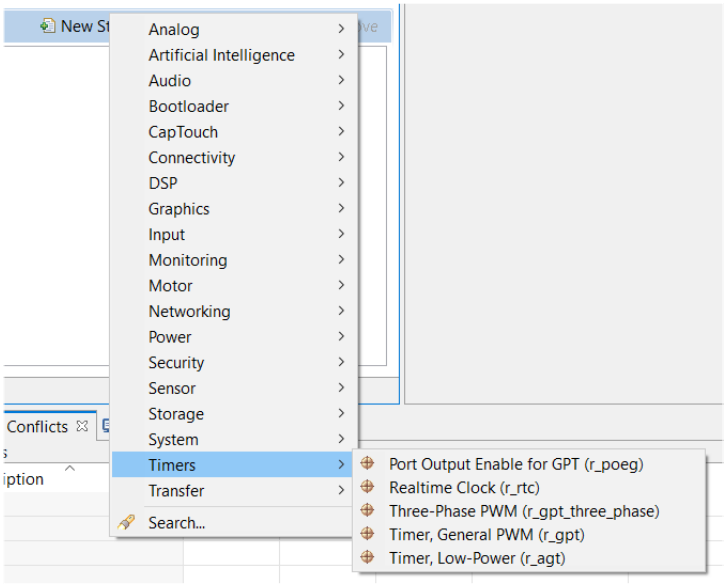



Figure 34: Select a driver

4. Select the driver module in the **HAL/Common Modules** pane and configure the driver properties in the **Properties** view.

e² studio adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the ra/fsp directory
- The main() function and configuration structures and header files for your application as shown in the table below.

| File | Contents | Overwritten by Generate Project Content? |
|---|---|---|
| ra_gen/main.c | Contains main() calling generated and user code. When called, the BSP already has Initialized the MCU. | Yes |
| ra_gen/hal_data.c | Configuration structures for HAL Driver only modules. | Yes |
| ra_gen/hal_data.h | Header file for HAL driver only modules. | Yes |
| src/hal_entry.c | User entry point for HAL Driver only code. Add your code here. | No |

The configuration header files for all included modules are created or overwritten in this folder: ra_cfg/fsp_cfg

#### 3.2.6.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the RTOS, you can add one or more threads, and for each thread at least one module that runs in the thread. You can select modules from the Driver dropdown menu. To add modules to a thread, follow these steps:

1. In the **Threads** pane, click **New Thread** to add a Thread.

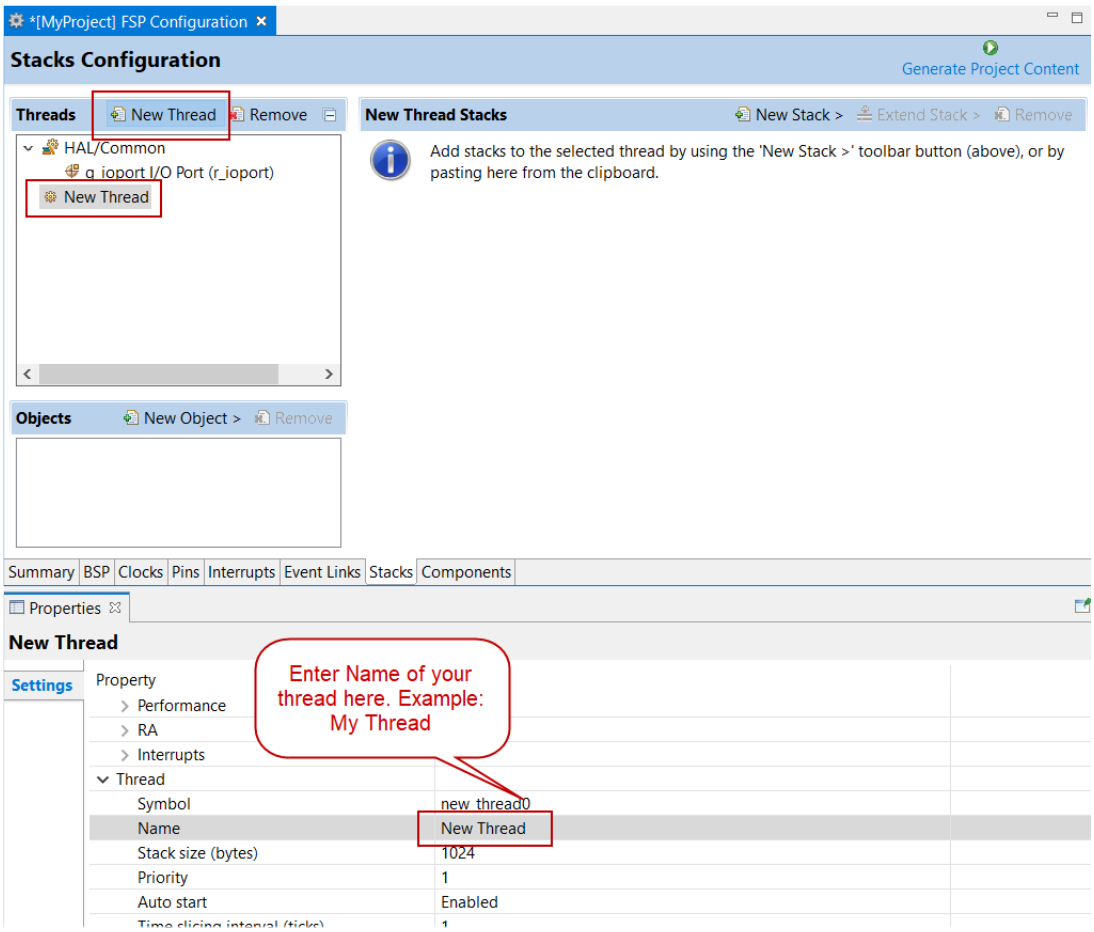



Figure 35: Adding a new RTOS Thread on the Stacks tab

2. In the **Properties** view, click on the **Name** and **Symbol** entries and enter a distinctive name and symbol for the new thread.

   *Note*

   *e² studio updates the name of the thread stacks pane to **My Thread Stacks**.*

3. In the **My Thread Stacks** pane, click on **New Stack** to see a list of modules and drivers. HAL-level drivers can be added here.

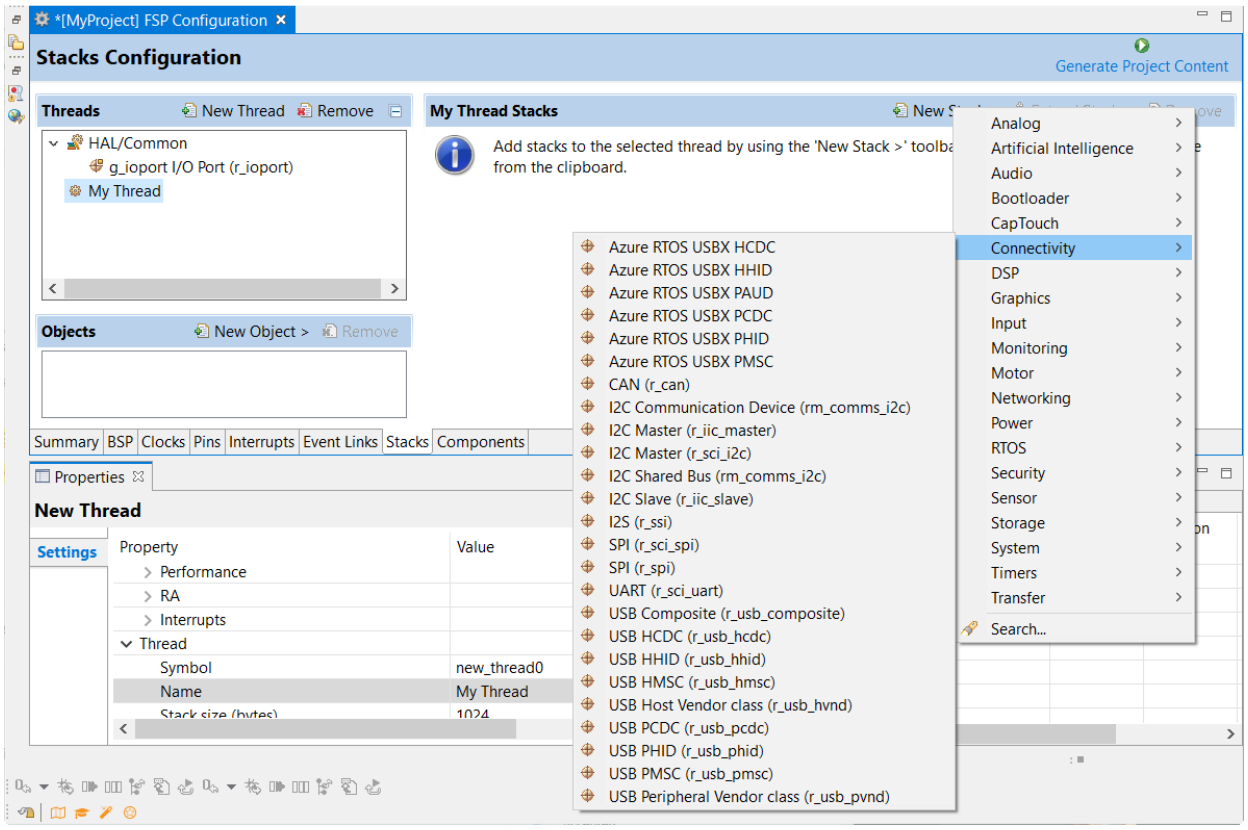



Figure 36: Adding Modules and Drivers to a thread

4. Select a module or driver from the list.

5. Click on the added driver and configure the driver as required by the application by updating the configuration parameters in the **Properties** view. To see the selected module or driver and be able to edit its properties, make sure the Thread containing the driver is highlighted in the **Threads** pane.

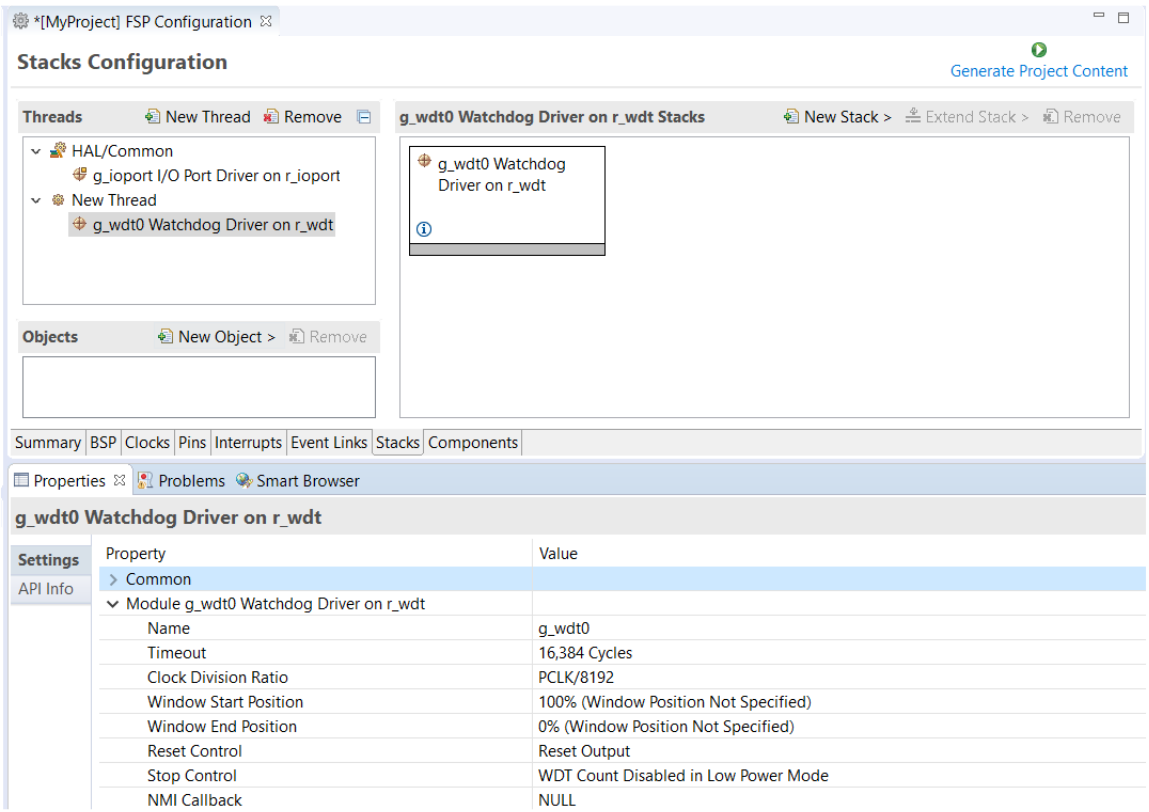



Figure 37: Configuring Module or Driver properties

6. If needed, add another thread by clicking **New Thread** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, e² studio creates the files as shown in the following table:

| File | Contents | Overwritten by Generate Project Content? |
|---|---|---|
| ra_gen/main.c | Contains main() calling generated and user code. When called the BSP will have initialized the MCU. | Yes |
| ra_gen/my_thread.c | Generated thread "my_thread" and configuration structures for modules added to this thread. | Yes |
| ra_gen/my_thread.h | Header file for thread "my_thread" | Yes |
| ra_gen/hal_data.c | Configuration structures for HAL Driver only modules. | Yes |
| ra_gen/hal_data.h | Header file for HAL Driver only modules. | Yes |
| src/hal_entry.c | User entry point for HAL Driver only code. Add your code here. | No |

| | | |
|---|---|---|
| src/my_thread_entry.c | User entry point for thread "my_thread". Add your code here. | No |

The configuration header files for all included modules and drivers are created or overwritten in the following folders: ra_cfg/fsp_cfg/<header files>

### 3.2.6.3 Configuring Threads

If the application uses an RTOS, the **Stacks** tab can be used to simplify the creation of RTOS threads, semaphores, mutexes, and event flags.

The components of each thread can be configured from the **Properties** view as shown below.

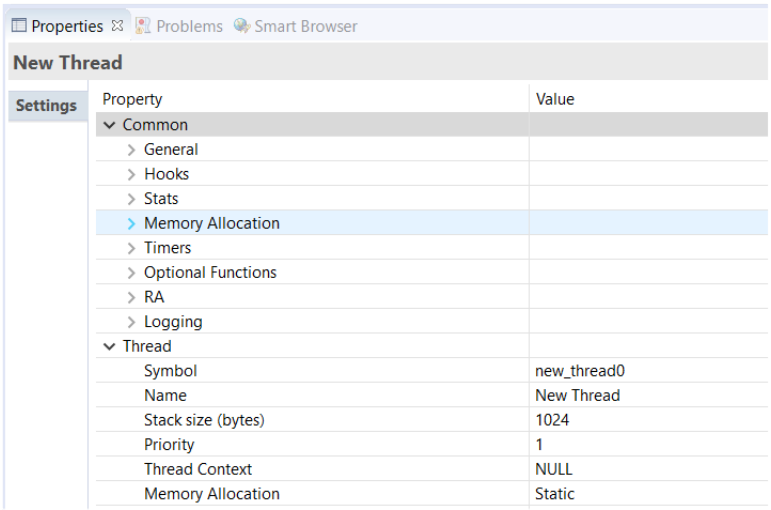



Figure 38: New Thread Properties

The **Properties** view contains settings common for all Threads (**Common**) and settings for this particular thread (**Thread**).

For this thread instance, the thread's name and properties (such as priority level or stack size) can be easily configured. e² studio checks that the entries in the property field are valid. For example, it will verify that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add RTOS resources to a Thread, select a thread and click on **New Object** in the Thread Objects pane. The pane takes on the name of the selected thread, in this case **My Thread Objects**.

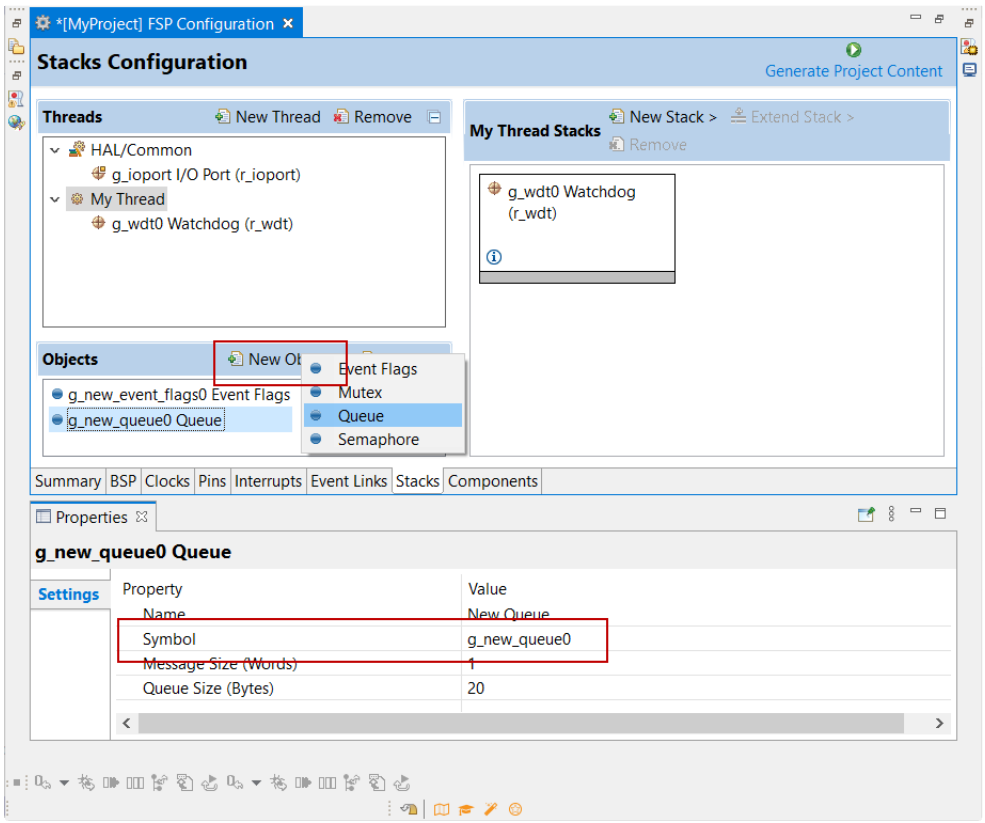



Figure 39: Configuring Thread Object Properties

Make sure to give each thread object a unique name and symbol by updating the **Name** and **Symbol** entries in the **Properties** view.

## 3.2.7 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all RA MCU projects are preselected (for example: **BSP > BSP > Board-specific BSP** and **HAL Drivers > all > r_cgc**). All modules that are necessary for the modules selected in the **Stacks** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.

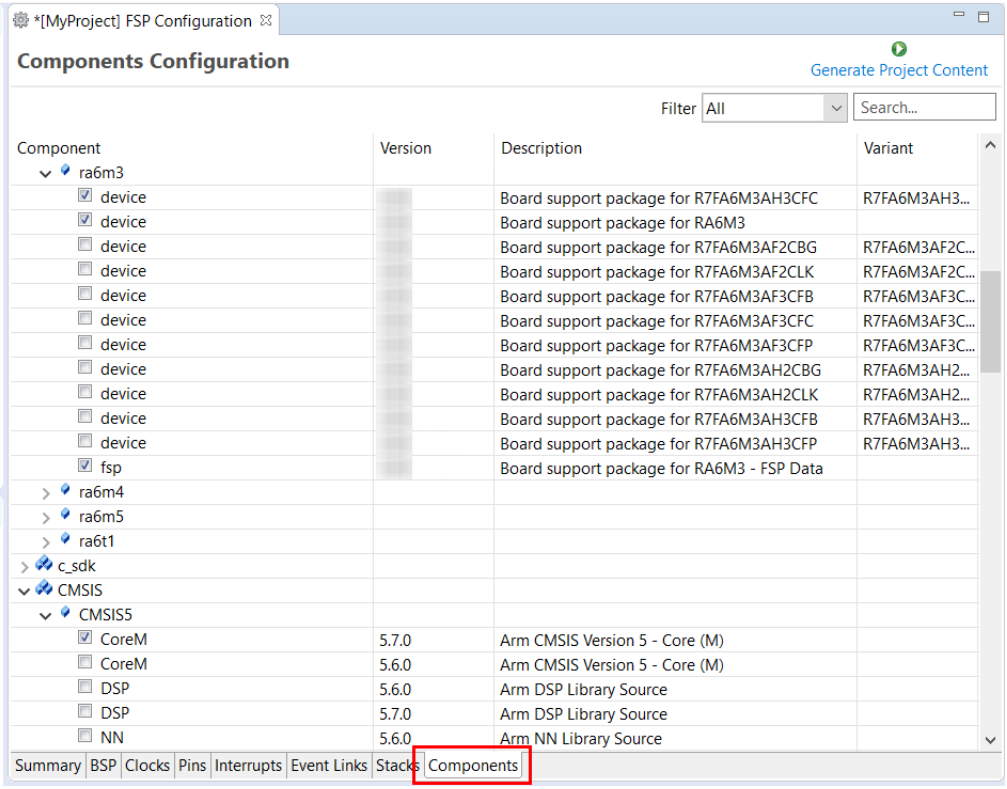



Figure 40: Components Tab

Clicking the **Generate Project Content** button copies the .c and .h files for each selected component into the following folders:

- ra/fsp/inc/api
- ra/fsp/inc/instances
- ra/fsp/src/bsp
- ra/fsp/src/<Driver_Name>

e² studio also creates configuration files in the ra_cfg/fsp_cfg folder with configuration options set in the **Stacks** tab.

## 3.2.8 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the **Stacks** tab, you can add the application code that calls the Modules and drivers.

*Note*

*To check your configuration, build the project once without errors before adding any of your own application code.*

### 3.2.8.1 Coding Features

e² studio provides several efficiency improving features that help write code. Review these features prior to digging into the code development step-by-step sections that follow.

## Autocomplete

Autocomplete is a context aware coding accelerator that suggests possible completions for partially

typed-in code elements. If you can 'guess' the first part of a macro, for example, the Autocomplete function can suggest options for completing the rest of the macro.

In the following example, a macro related to a BSP_IO setting needs to be found. After typing BSP_IO_ in a source code file, pressing Ctrl + Space opens the Autocomplete list. This list shows a selection of context aware options for completing the macro. Scroll through the window to find the desired macro (in this case BSP_IO_LEVEL_HIGH) and click on it to add it to your code.

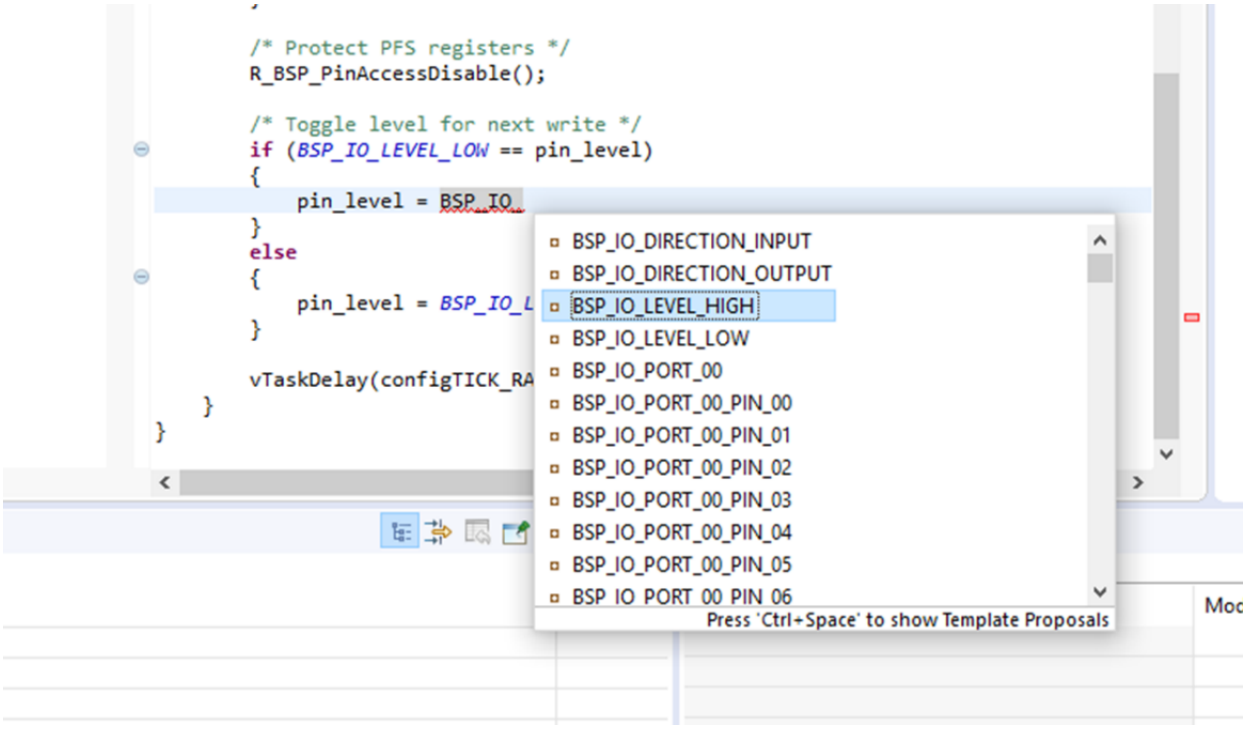



Figure 41: Autocomplete example

Other code elements can use autocomplete too. Some of the more common uses for Autocomplete include Enumerations, Types, and API functions - but try it in any situation you think the tool may have enough context to determine what you might be looking for.

For a hands-on experience using Autocomplete use the Quick FSP Labs for Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

### Welcome Window

The e² studio Welcome window displays useful information and common links to assist in development. Check out these resources to see what is available. They are updated with each release, so check back to see what has been added after a new release.

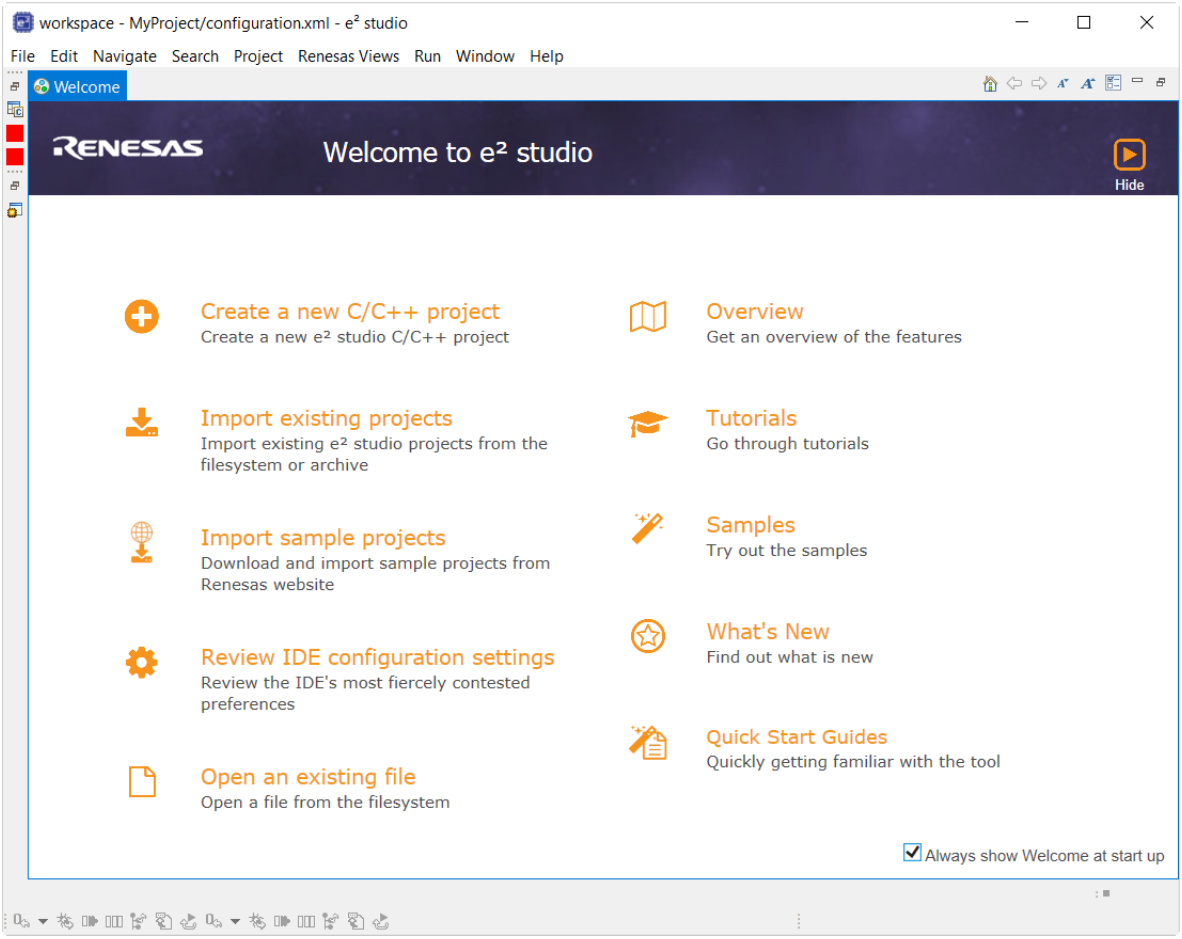



Figure 42: Welcome window

## Cheat Sheets

Cheat sheets are macro driven illustrations of some common tasks. They show, step-by-step, what commands and menus are used. These will be populated with more examples on each release. Cheat Sheets are available from the **Help** menu.

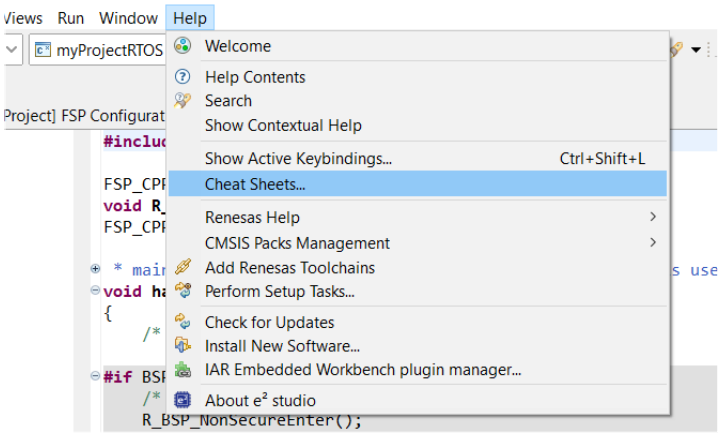



Figure 43: Cheat Sheets

## Developer Assistance

FSP Developer Assistance provides developers with module and Application Programming Interface (API) reference documentation in e² studio. After configuring the threads and software stacks for an FSP project with the RA Configuration editor, Developer Assistance quickly helps you get started writing C/C++ application code for the project using the configured stack modules.

1. Expand the project explorer to view Developer Assistance

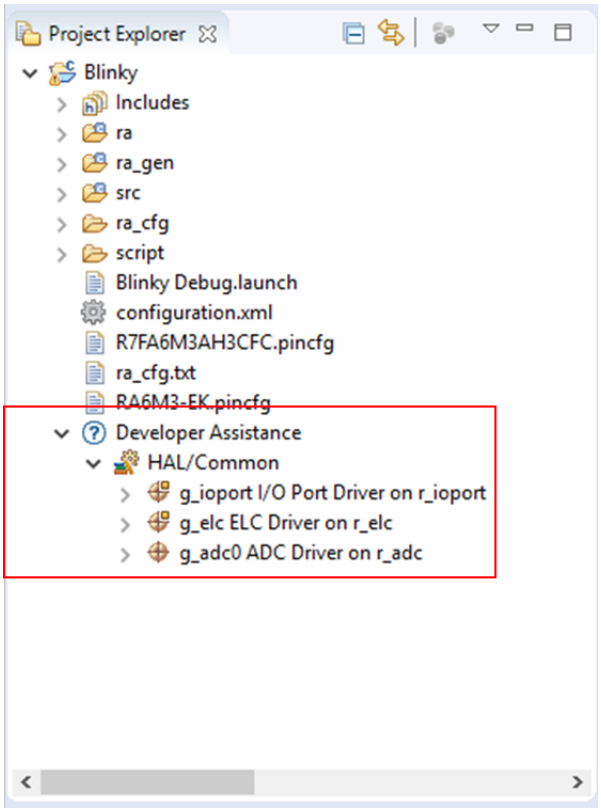



Figure 44: Developer Assistance

2. Expand a stack module to show its APIs

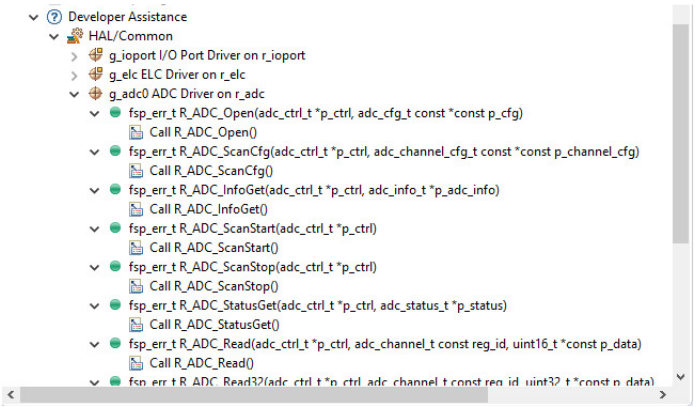



Figure 45: Developer Assistance APIs

3. Dragging and dropping an API from Develop Assistance to a source file helps to write source code quickly.

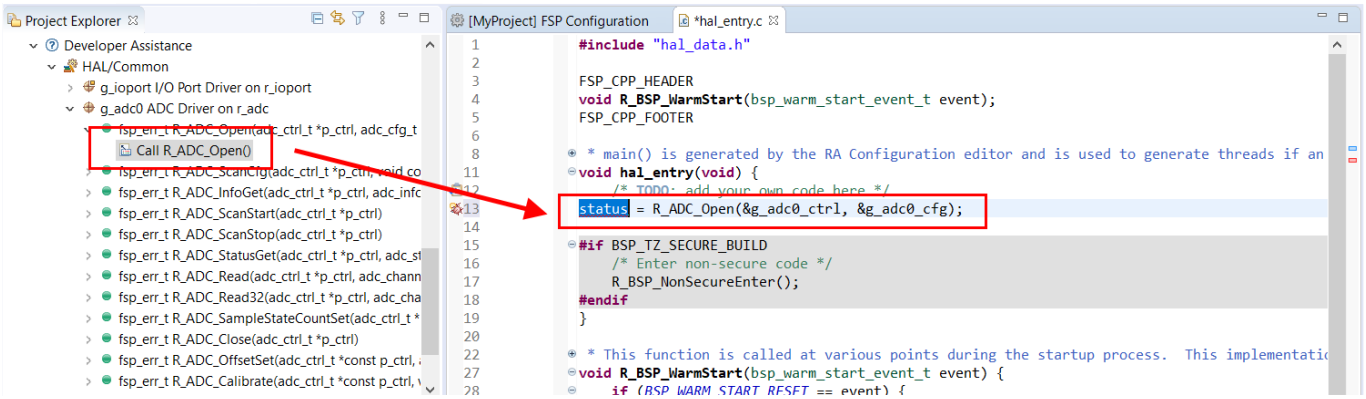



Figure 46: Dragging and Dropping an API in Developer Assistance

For a hands-on experience using Developer Assistance use the Quick FSP Labs for An Introduction to Developer Assistance, Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

### Information Icon

Information icons are available on each module in the thread stack. Clicking on these icons opens a module folder on GitHub that contains additional information on the module. An example information Icon is shown below:

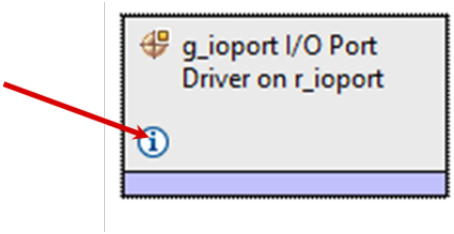



Figure 47: Information icon

### IDE Help

A good source of additional information for many FSP topics is the Help system. To get to the Help system, click on **Help** and then select **Help Contents** as seen below.

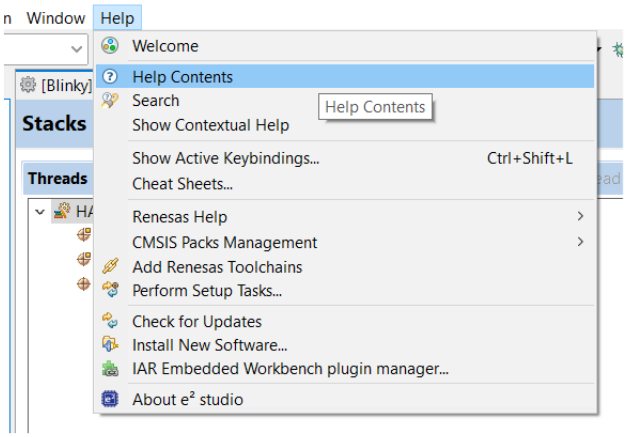



Figure 48: Opening the Help System

Once the Help system is open, select the **RA Contents** entry in the left side Guide-bar. Expand it to see the main RA Topics.

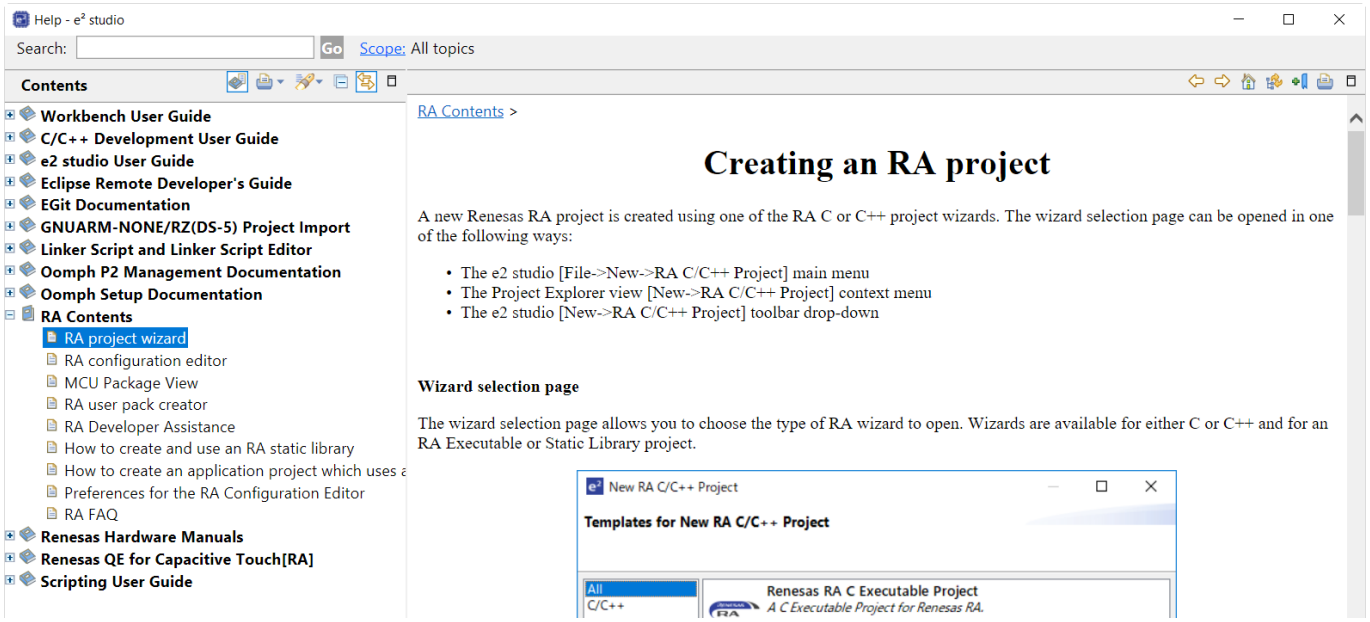



Figure 49: RA Content Help

You can also search for help topics by using the Search bar. Below is an example searching for Visual Expressions, a helpful feature in the e² studio debugger.

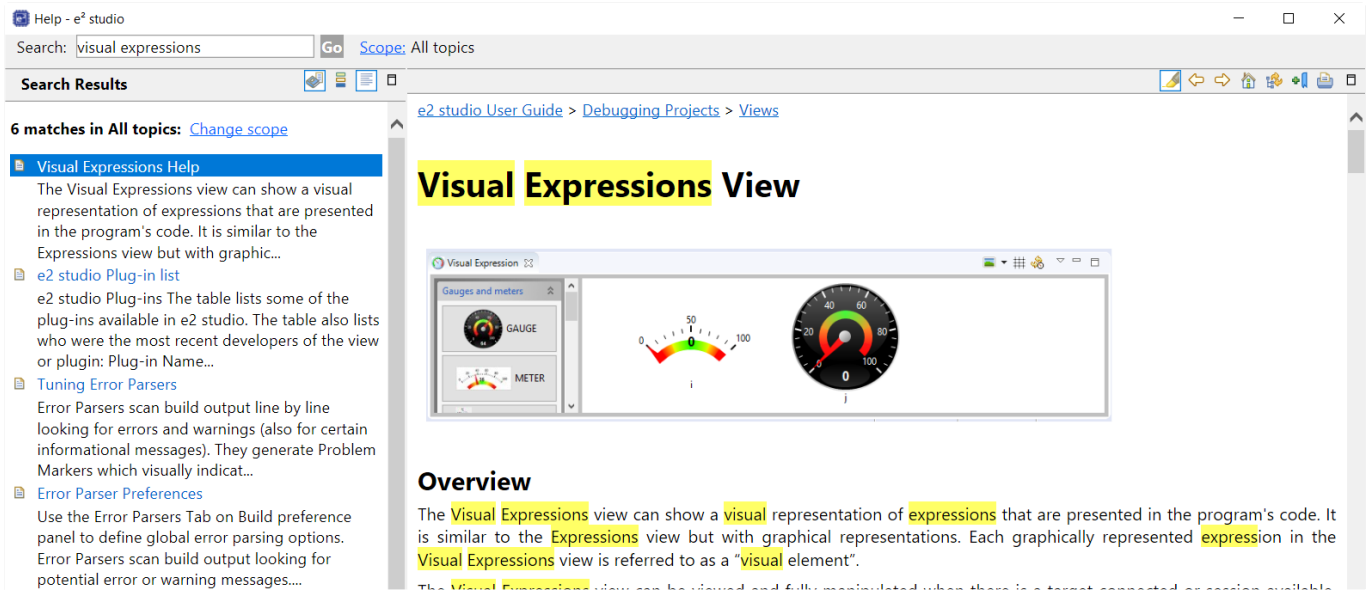



Figure 50: e² studio Help from the Search Bar

For a hands-on experience using the Help system use the Quick FSP Labs for An Introduction to Developer Assistance, Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

### 3.2.8.2 HAL Modules in FSP: A Practical Description

The FSP Architecture section describes FSP stacks, modules and interfaces in significant detail, providing an understanding of the theory behind them. The following sections provides a quick and practical introduction on how to use API functions when writing code and where in the API reference sections you can find useful API related information.

#### Introduction to HAL Modules

In FSP, HAL module drivers provide convenient API functions that access RA processor peripheral features. Module properties are defined in the RA GUI configurator, eliminating the tedious and error prone process of setting peripheral control registers. When configuration is complete, the generator automatically creates the code needed to implement the associated API functions. API functions are the main way a developer interacts with the target processor and peripherals.

#### HAL Driver API Function Call Formats

HAL driver API functions all have a similar format. They all start with "R_" to indicate they are HAL related functions. Next comes the module name followed by the function and any parameters. This format is illustrated below:

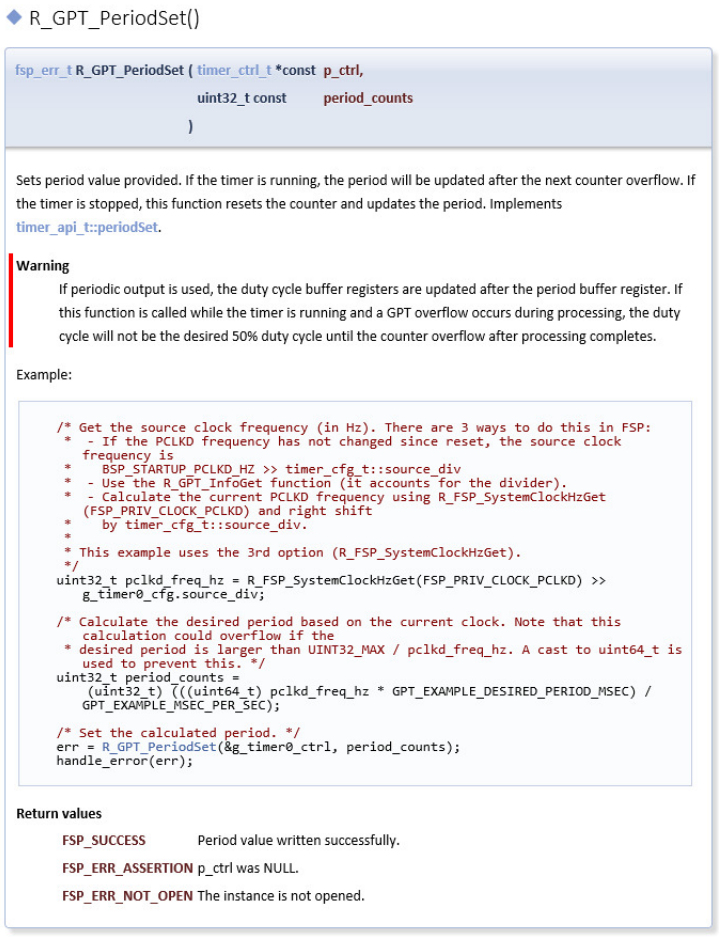
```
R_<module>_<function>(<parameters>);
```

Here are some examples:

```
status = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
status = R_GPT_Start(&g_timer0_ctrl);
status = R_GPT_PeriodSet(&g_timer0_ctrl, period);
status = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
status = R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
```

### HAL Driver API Call Reference Information

Each HAL module has a useful API Reference section that includes key details on each function. The function prototype is presented first, showing the return type (usually fsp_status_t for HAL functions) and the function parameters. A short description and any warnings or notes follow the function definition. In some cases, a code snippet is included to illustrate use of the function. Finally, all possible return values are provided to assist in debugging and error management.



◆ R_GPT_PeriodSet()

```
fsp_err_t R_GPT_PeriodSet ( timer_ctrl_t *const  p_ctrl,
                            uint32_t const       period_counts
                          )
```

Sets period value provided. If the timer is running, the period will be updated after the next counter overflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and a GPT overflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter overflow after processing completes.

Example:

```
/* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
 *  - If the PCLKD frequency has not changed since reset, the source clock
   frequency is
 *    BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
 *  - Use the R_GPT_InfoGet function (it accounts for the divider).
 *  - Calculate the current PCLKD frequency using R_FSP_SystemClockHzGet
   (FSP_PRIV_CLOCK_PCLKD) and right shift
 *    by timer_cfg_t::source_div.
 *
 * This example uses the 3rd option (R_FSP_SystemClockHzGet).
 */
uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
   g_timer0_cfg.source_div;

/* Calculate the desired period based on the current clock. Note that this
   calculation could overflow if the
 * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
   used to prevent this. */
uint32_t period_counts =
    (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
   GPT_EXAMPLE_MSEC_PER_SEC);

/* Set the calculated period. */
err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
handle_error(err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

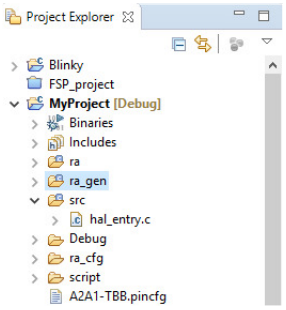
Figure 51: Module Api Reference Section Example

### 3.2.8.3 RTOS-Independent Applications

To write application code:

1. Add all drivers and modules in the **Stacks** tab and resolve all dependencies flagged by e² studio such as missing interrupts or drivers.
2. Configure the drivers in the **Properties** view.
3. In the Project Configuration view, click the **Generate Project Content** button.

4. In the **Project Explorer** view, double-click on the src/hal_entry.c file to edit the source file.




*Note*

*All configuration structures necessary for the driver to be called in the application are initialized in ra_gen/hal_data.c.*

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

5. Add your application code here:

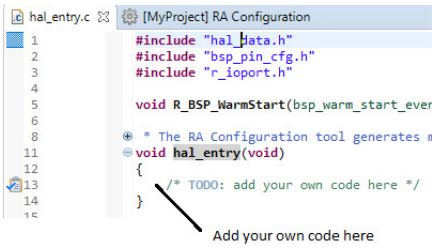



Figure 52: Adding user code to hal_entry.c

6. Build the project without errors by clicking on **Project > Build Project**.

The following tutorial shows how execute the steps above and add application code: Tutorial: Using HAL Drivers - Programming the WDT.

The WDT example is a HAL level application which does not use an RTOS. The user guides for each module also include basic application code that you can add to hal_entry.c.

### 3.2.8.4 RTOS Applications

To write RTOS-aware application code using RTOS, follow these steps:

1. Add a thread using the **Stacks** tab.
2. Provide a unique name for the thread in the **Properties** view for this thread.

3. Configure all drivers and resources for this thread and resolve all dependencies flagged by e² studio such as missing interrupts or drivers.
4. Configure the thread objects.
5. Provide unique names for each thread object in the **Properties** view for each object.
6. Add more threads if needed and repeat steps 1 to 5.
7. In the **RA Project Editor**, click the **Generate Project Content** button.

8. In the **Project Explorer** view, double-click on the src/my_thread_1_entry.c file to edit the source file.

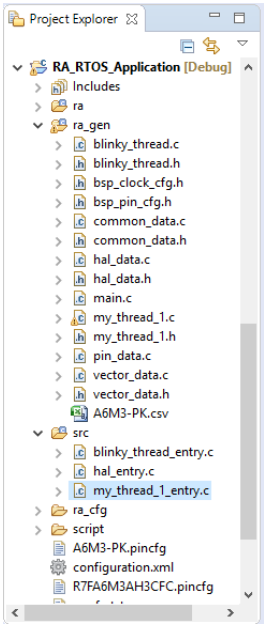



Figure 53: Generated files for an RTOS application

*Note*

*All configuration structures necessary for the driver to be called in the application are initialized in ra_gen/my_thread_1.c and my_thread_2.c*

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

9. Add your application code here:

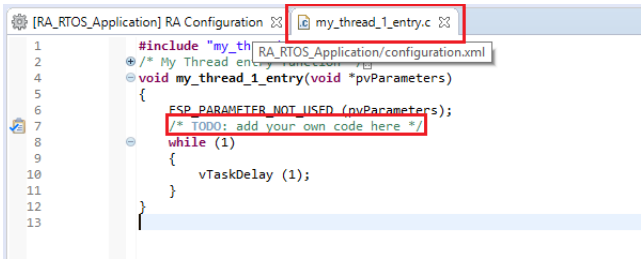



Figure 54: Adding user code to my_thread_1.entry

10. Repeat steps 1 to 9 for the next thread.

11. Build your project without errors by clicking on **Project > Build Project**.

### 3.2.8.5 Additional Resources for Application Development

#### Example Projects

A wide variety of Example Projects for FSP and RA MCUs is available on the GitHub site here: https://github.com/renesas/ra-fsp-examples. Example projects are organized by target kit so it is easy to find all the examples for your kit of choice.

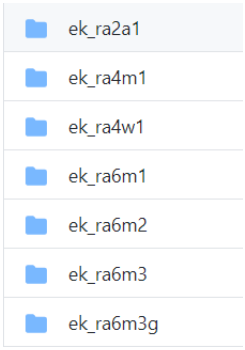



Figure 55: FSP Example Projects Organized by Kit

Projects are available as both downloadable zip files and as project source files. Typically, there is a project for each module. New example projects are being added periodically, so check back if a particular module isn't yet available.

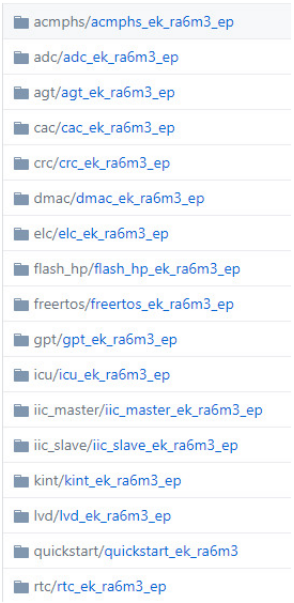



Figure 56: A Selection of Example Projects Available on GitHub

#### Quick Labs

A variety of Hands-on Do It Yourself labs are available on the Renesas RA and FSP Knowledge Base. Quick FSP Labs target the EK-RA6M3 kit and typically require only 15 minutes to complete. Each lab covers a couple related development tools and techniques like Autocomplete, Developer Assistance,

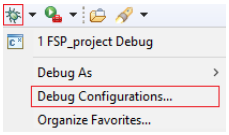
console I/O over RTT, and Visual Expressions, that can speed up the development process. A list of all available Quick Labs can be found here: https://en-support.renesas.com/knowledgeBase/19450948

## 3.2.9 Debugging the Project

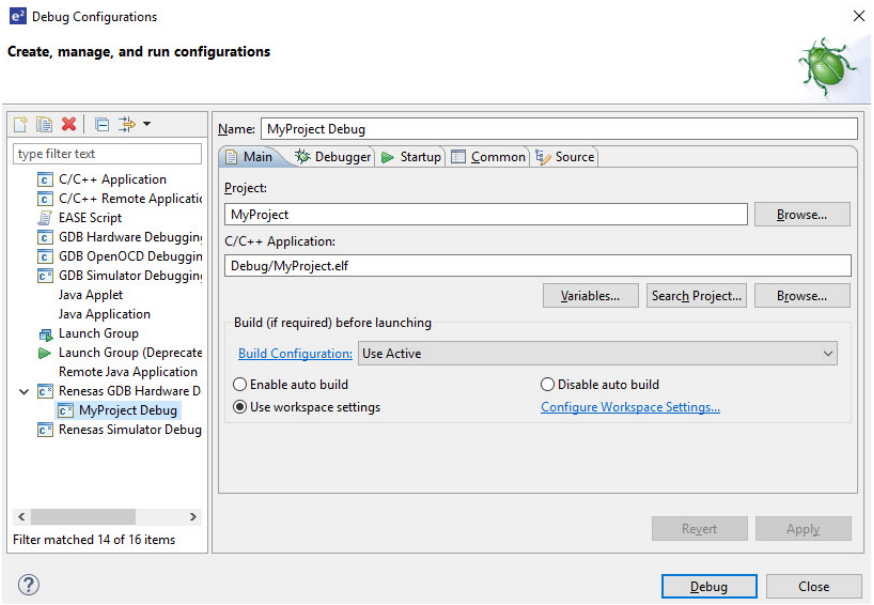
Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

1. On the drop-down list next to the debug icon, select **Debug Configurations**.




2. In the **Debug Configurations** view, click on your project listed as **MyProject Debug**.




3. Connect the board to your PC via either a standalone Segger J-Link debugger, a Segger J-Link On-Board (included on all RA EKs), or an E2 or E2 Lite and click **Debug**.

*Note*

*For details on using J-Link and connecting the board to the PC, see the Quick Start Guide included in the RA MCU Kit.*

## 3.2.10 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within e² studio through the menu **Project > Properties > C/C++ Build > Settings** when the project is selected. The following screenshots show the settings dialogs for the GNU Arm and LLVM toolchains. The dialog looks slightly different depending upon the toolchain being used.

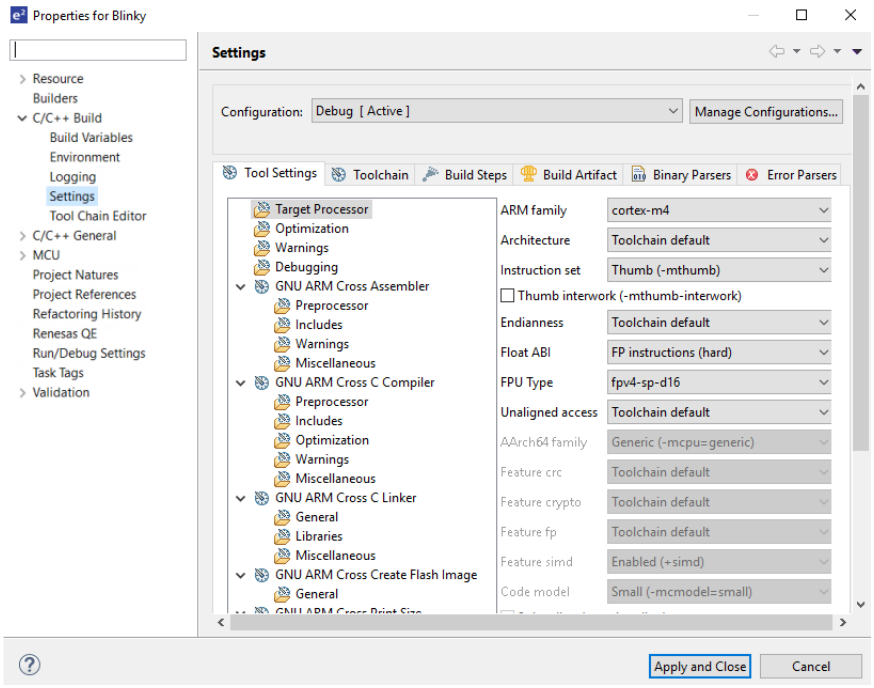



Figure 57: e² studio Project toolchain settings for GNU Arm

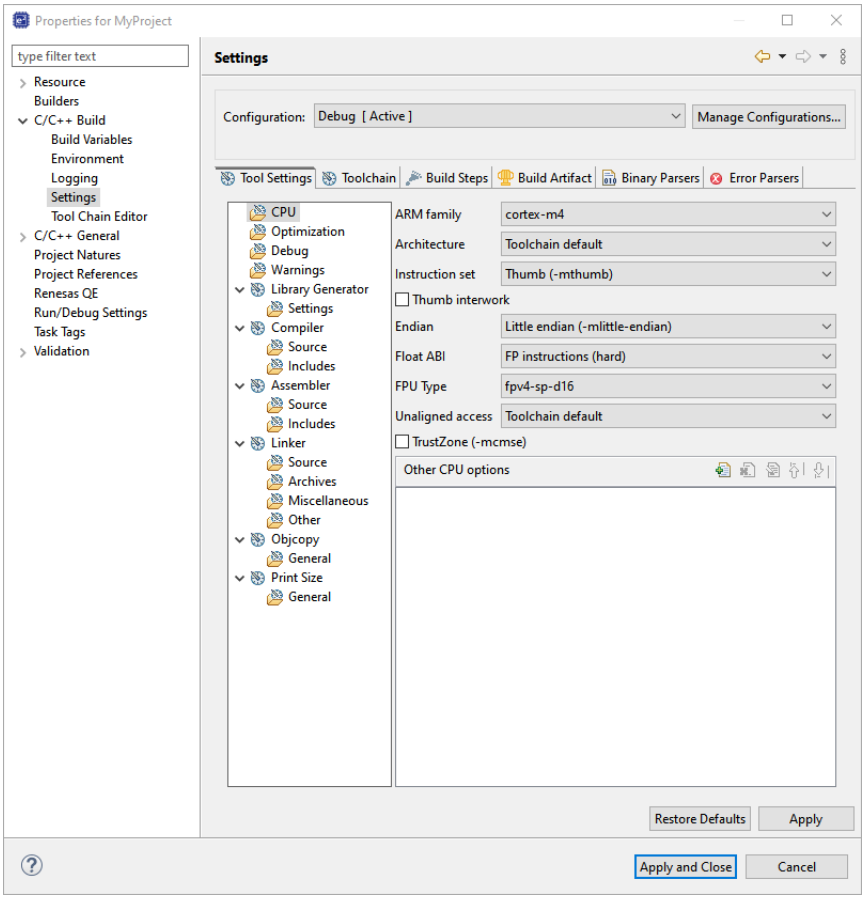



Figure 58: e² studio Project toolchain settings for LLVM

The scope for the settings is project scope which means that the settings are valid only for the project being modified.

The settings for the linker which control the location of the various memory sections are contained in a script file specific for the device being used. This script file is included in the project when it is created and is found in the script folder (for example, /script/a6m3.ld).

## 3.2.11 Creating RA project with Arm Compiler 6 in e² studio

e² studio does not include the Arm Compiler 6 (AC6) toolchain by default. Follow the steps below to integrate AC6 into e² studio and create an AC6 RA project.

*Note*

*It is assumed that the user is already familiar with RA project creation in e² studio. e² studio does not include Arm Compiler 6 (AC6) toolchain by default.*

Steps 1 through 8 describe the process for integrating Arm Compiler 6 into e² studio.

1. Download, install, and configure license for the AC6 toolchain (https://developer.arm.com/tools-and-software/embedded/arm-compiler/downloads/version-6).
2. Launch e² studio.
3. Go to **Window > Preferences > Toolchains**.

4. Click **Add**.

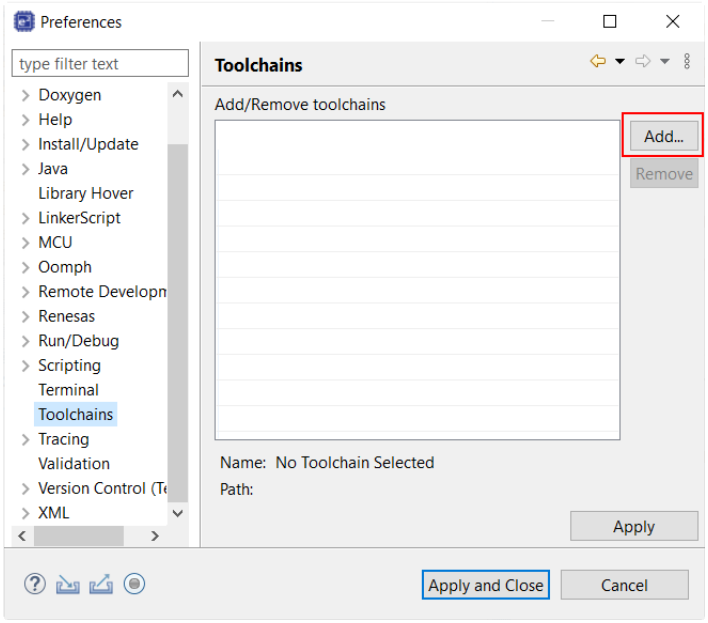



Figure 59: Add Toolchain

5. Browse to the path where AC6 toolchain is installed and select the \bin folder. Click **Next**.

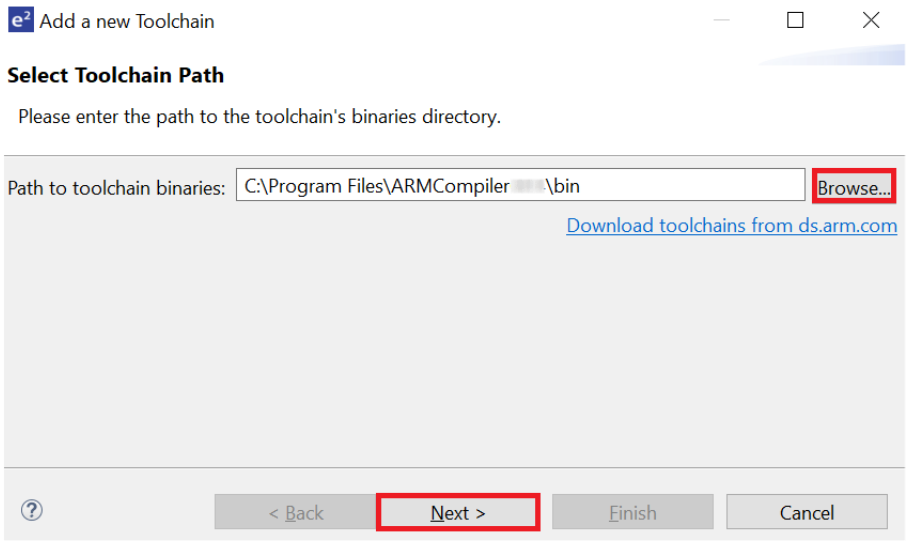



Figure 60: Browse to AC6 Compiler

6. Toolchain information in displayed. Click **Finish**.

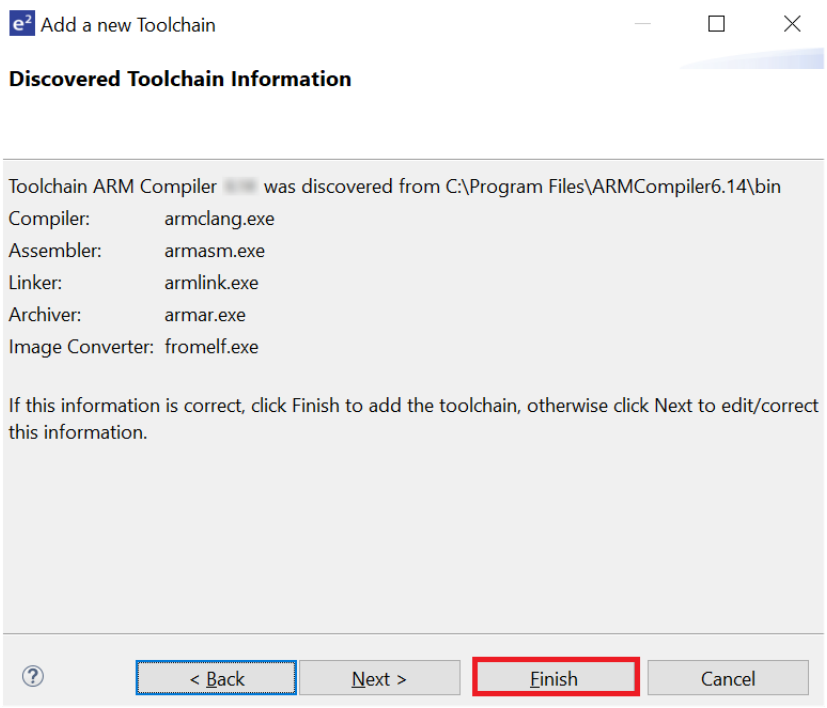



Figure 61: Toolchain Information

7. Click **Apply and Close**.

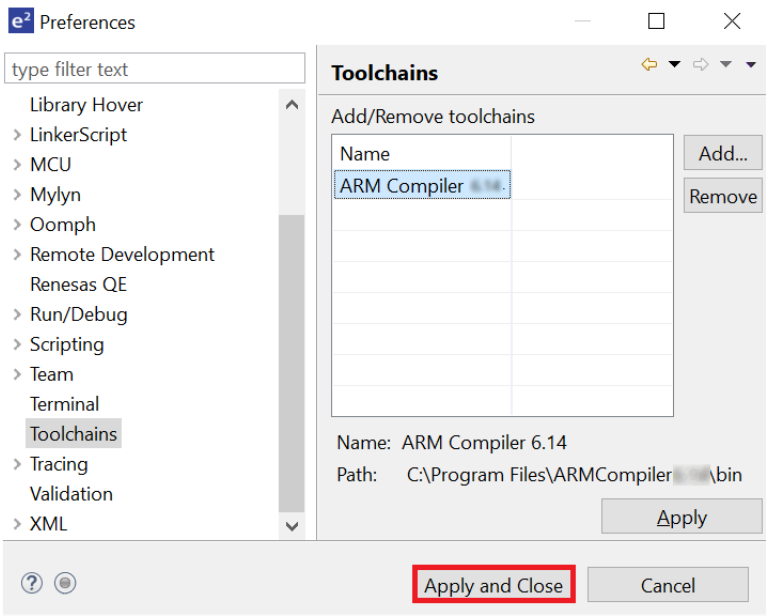



Figure 62: Apply and Close

8. Click **Restart Eclipse** when prompted.

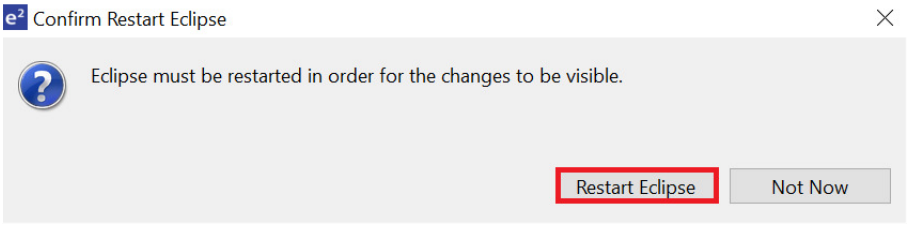



Figure 63: Restart Eclipse

9. When creating a new RA C/C++ project, select **ARM Compiler 6** included in the Toolchains section.

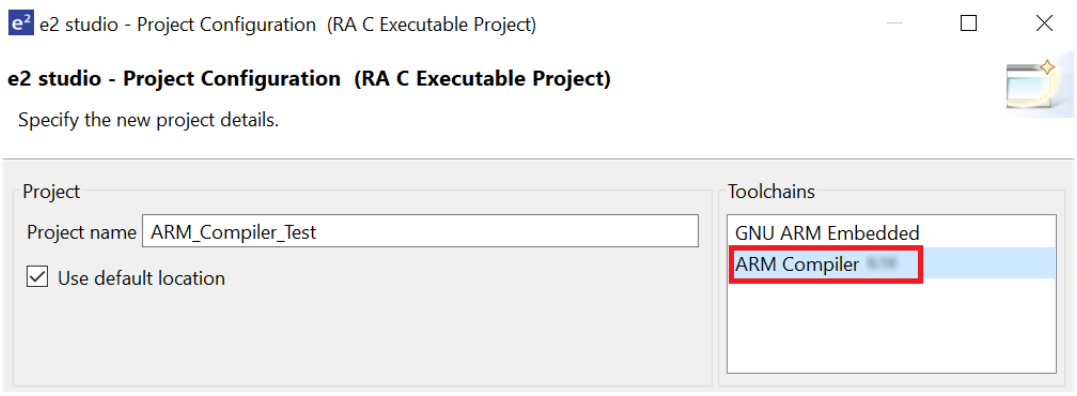



Figure 64: Select Arm Compiler

## 3.2.12 Importing an Existing Project into e² studio

1. Start by opening e² studio.
2. Open an existing Workspace to import the project and skip to step d. If the workspace doesn't exist, proceed with the following steps:

    a. At the end of e² studio startup, you will see the Workspace Launcher Dialog box as shown in the following figure.

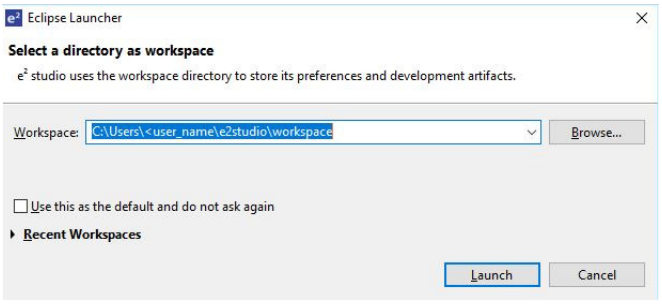



Figure 65: Workspace Launcher dialog

b. Enter a new workspace name in the Workspace Launcher Dialog as shown in the following figure. e² studio creates a new workspace with this name.

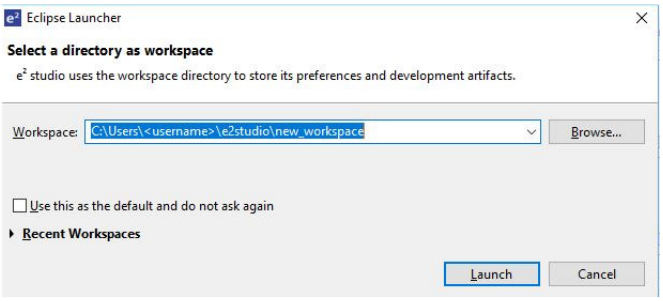



Figure 66: Workspace Launcher dialog - Select Workspace

c. Click **Launch**.

d. When the workspace is opened, you may see the Welcome Window. Click on the **Hide** arrow button to proceed past the Welcome Screen as seen in the following figure.

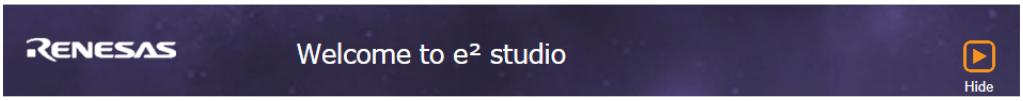



Figure 67: Workbench arrow button

3. You are now in the workspace that you want to import the project into. Click the **File** menu in the menu bar, as shown in the following figure.

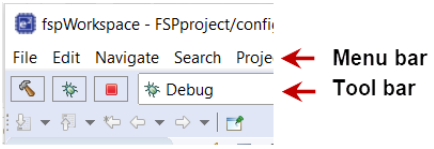



Figure 68: Menu and tool bar

4. Click **Import** on the **File** menu or in the menu bar, as shown in the following figure.

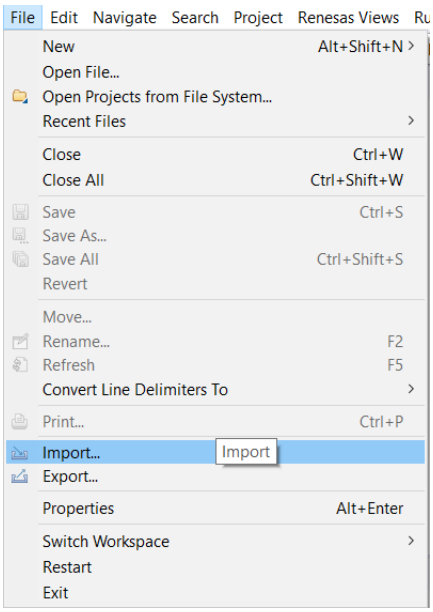



Figure 69: File drop-down menu

5. In the **Import** dialog box, as shown in the following figure, choose the **General** option, then **Existing Projects into Workspace**, to import the project into the current workspace.

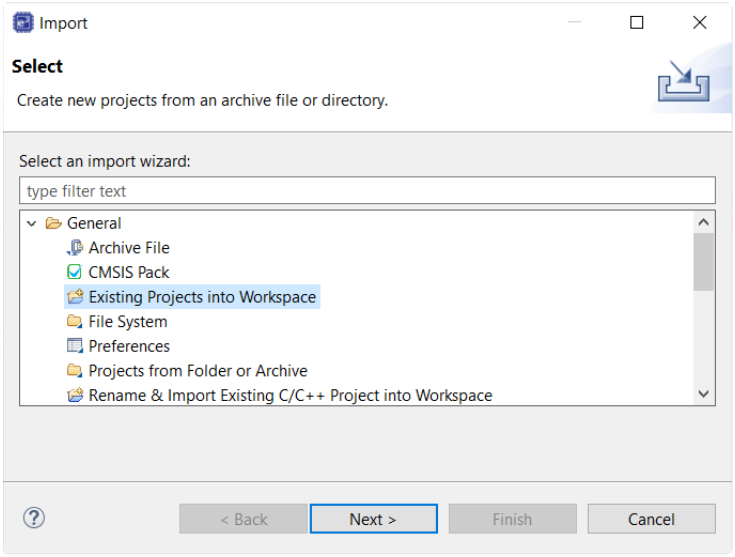



Figure 70: Project Import dialog with "Existing Projects into Workspace" option selected

6. Click **Next**.
7. To import the project, use either **Select archive file** or **Select root directory**.

    a. Click **Select archive file** as shown in the following figure.

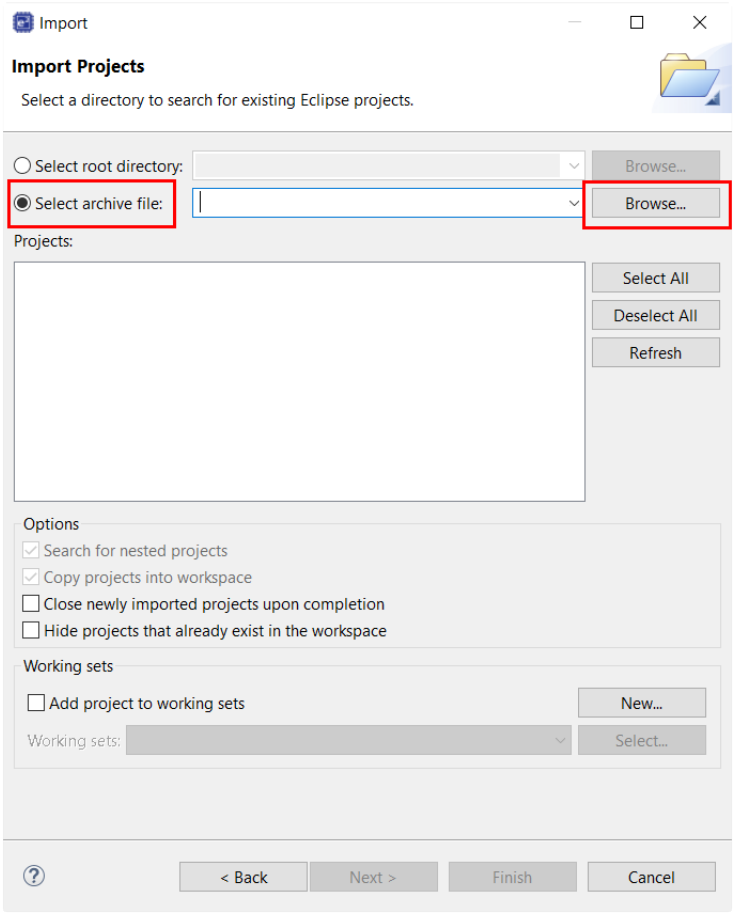



Figure 71: Import Existing Project dialog 1 - Select archive file

b. Click **Select root directory** as shown in the following figure.

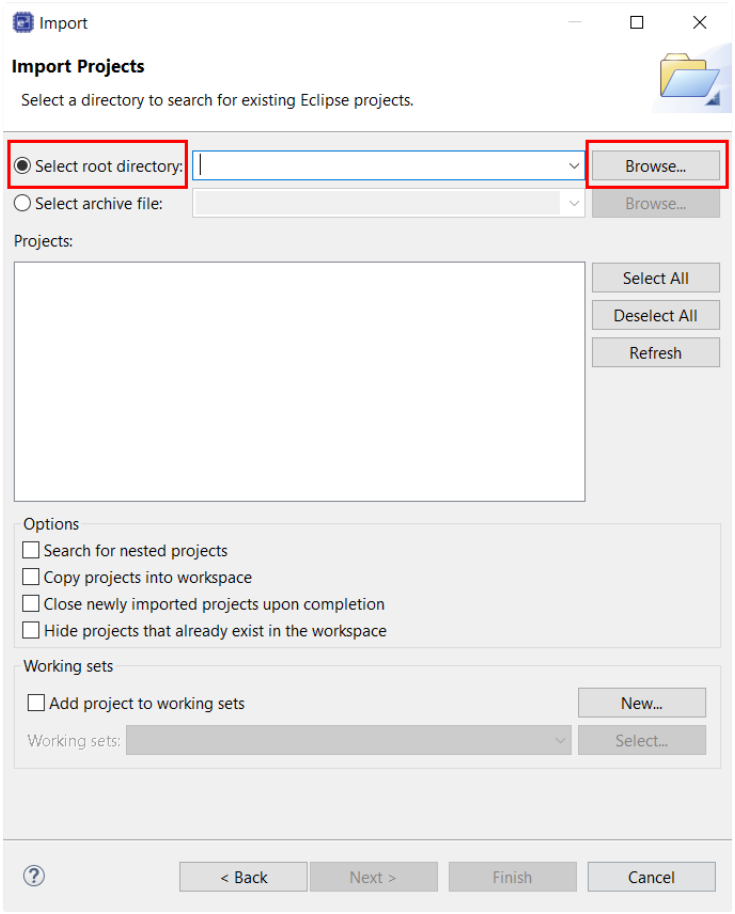



Figure 72: Import Existing Project dialog 1 - Select root directory

8. Click **Browse**.
9. For **Select archive file**, browse to the folder where the zip file for the project you want to import is located. For **Select root directory**, browse to the project folder that you want to import.
10. Select the file for import. In our example, it is CAN_HAL_MG_AP.zip or CAN_HAL_MG_AP.
11. Click **Open**.

12. Select the project to import from the list of **Projects**, as shown in the following figure.

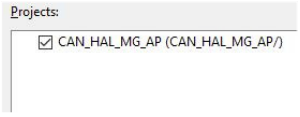



Figure 73: Import Existing Project dialog 2

13. Click **Finish** to import the project.

## 3.2.13 Using Semihosting in a Project

*Note*

***printf requires use of the heap (BSP Tab -> Properties -> RA Common -> Heap size (bytes))***

When using certain standard C I/O functions such as printf and scanf semihosting must be initialized

for them to work correctly with the **Renesas Debug Virtual Console**. In order to setup semihosting a call to initialise_monitor_handles should be added somewhere in your application before any semihosting related calls are made. Here is an example declaration and call:

```
extern void initialise_monitor_handles(void); /* Add this declaration before calling.
*/

initialise_monitor_handles(); /* Add this call to your application. */
```

# 3.3 Tutorial: Your First RA MCU Project - Blinky

## 3.3.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the steps of creating a simple application using e² studio and running that application on an RA MCU board.

## 3.3.2 What Does Blinky Do?

The application used in this tutorial is Blinky, traditionally the first program run in a new embedded development environment.

Blinky is the "Hello World" of microcontrollers. If the LED blinks you know that:

- The toolchain is setup correctly and builds a working executable image for your chip.
- The debugger has installed with working drivers and is properly connected to the board.
- The board is powered up and its jumper and switch settings are probably correct.
- The microcontroller is alive, the clocks are running, and the memory is initialized.

The Blinky example application used in this tutorial is designed to run the same way on all boards offered by Renesas that hold the RA microcontroller. The code in Blinky is completely board independent. It does the work by calling into the BSP (board support package) for the particular board it is running on. This works because:

- Every board has at least one LED connected to a GPIO pin.
- That one LED is always labelled LED1 on the silk screen.
- Every BSP supports an API that returns a list of LEDs on a board, and their port and pin assignments.

## 3.3.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e² studio
- Flexible Software Package
- An RA MCU board kit

## 3.3.4 Create a New Project for Blinky

The creation and configuration of an RA MCU project is the first step in the creation of an application.

The base RA MCU pack includes a pre-written Blinky example application that is simple and works on all Renesas RA MCU boards.

Follow these steps to create an RA MCU project:

1. In e² studio, click **File > New > C/C++ Project > Renesas RA** and select **Renesas RA C/C++ Project**.
2. Assign a name to this new project. Blinky is a good name to use for this tutorial.

3. Click **Next**. The **Project Configuration** window shows your selection.

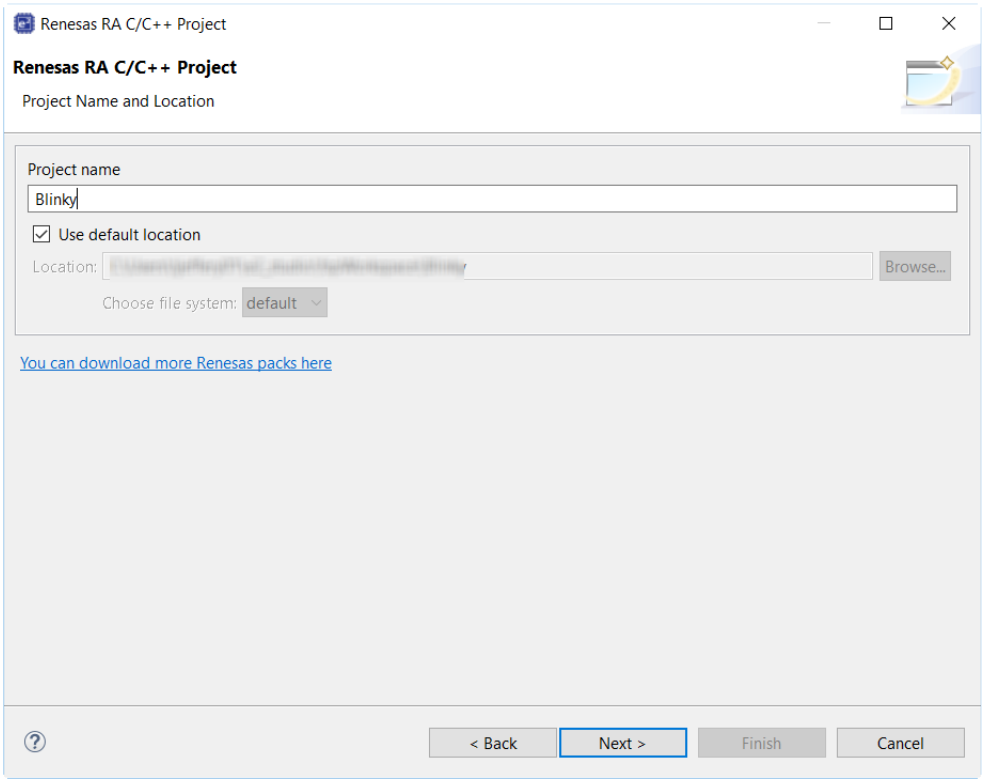



Figure 74: e² studio Project Configuration window (part 1)

4. Select the board support package by selecting the name of your board from the **Device Selection** drop-down list and click **Next**.

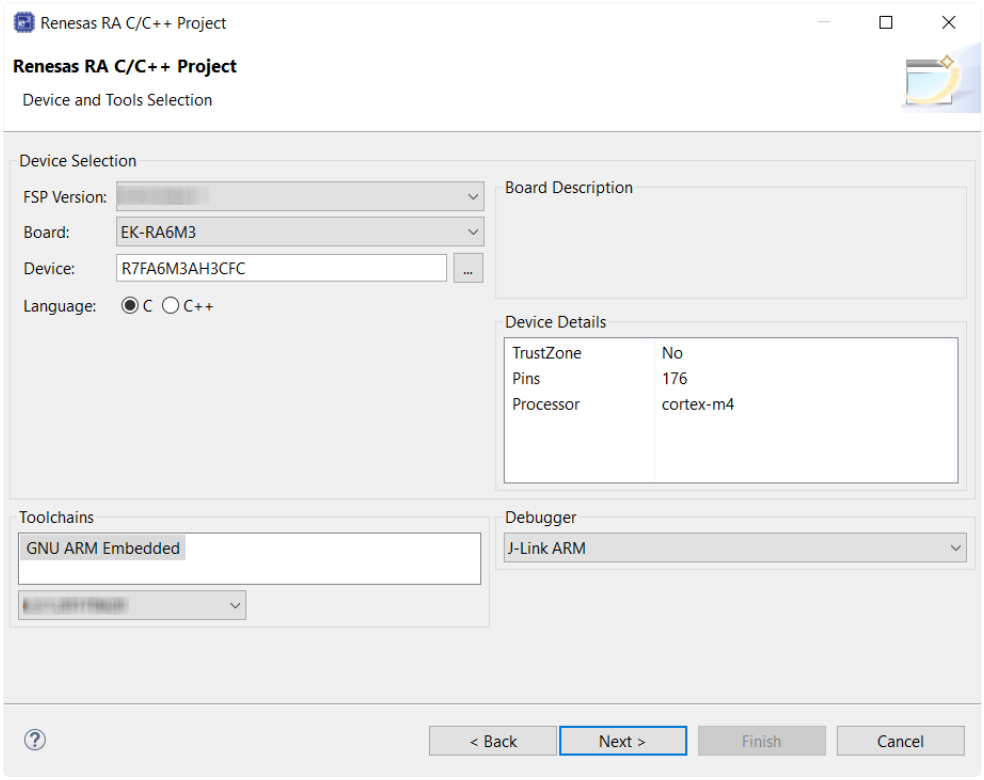



Figure 75: e² studio Project Configuration window (part 2)

5. Select the build artifact and RTOS.

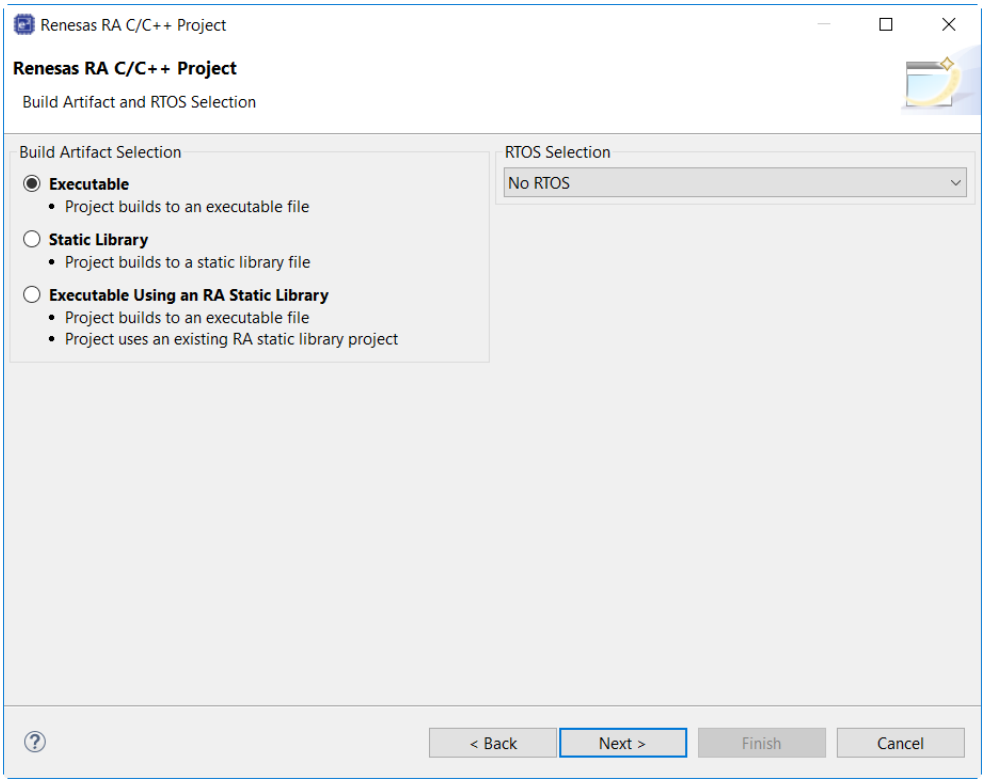



Figure 76: e² studio Project Configuration window (part 3)

6. Select the Blinky template for your board and click **Finish**.

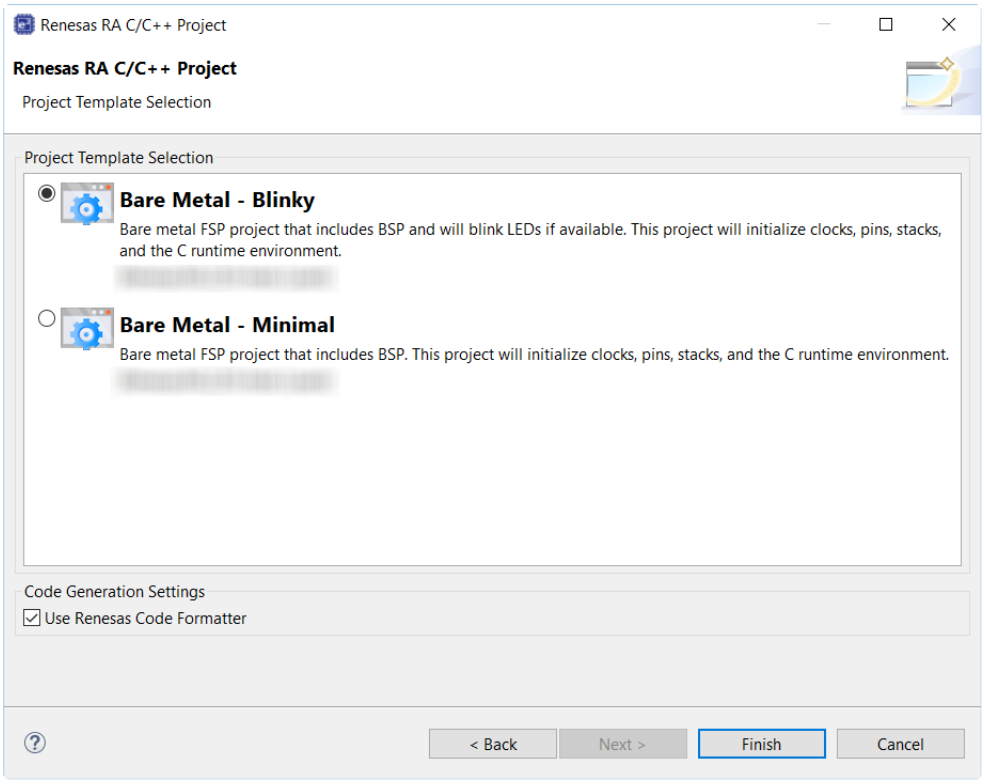



Figure 77: e² studio Project Configuration window (part 4)

Once the project has been created, the name of the project will show up in the **Project Explorer** window of e² studio. Now click the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

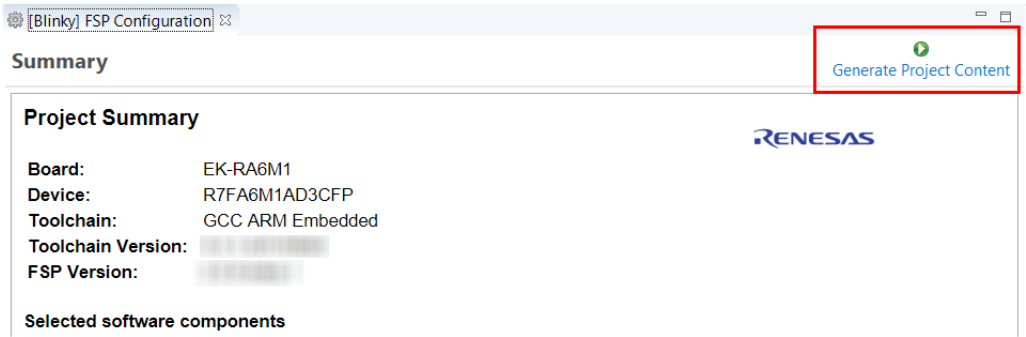



Figure 78: e² studio Project Configuration tab

Your new project is now created, configured, and ready to build.

### 3.3.4.1 Details about the Blinky Configuration

The **Generate Project Content** button creates configuration header files, copies source files from templates, and generally configures the project based on the state of the **Project Configuration** screen.

For example, if you check a box next to a module in the **Components** tab and click the **Generate Project Content** button, all the files necessary for the inclusion of that module into the project will be copied or created. If that same check box is then unchecked those files will be deleted.

#### 3.3.4.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by e² studio for the Blinky application. The clock configuration tab (see Configuring Clocks) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file (see BSP Clock Configuration).

#### 3.3.4.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by e² studio for the Blinky application. The pin configuration tab shows the pin configuration for the Blinky application (see Configuring Pins). The Blinky pin configuration is stored in the BSP configuration file (see BSP Pin Configuration).

#### 3.3.4.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the Components tab:

- r_ioport

To see the configuration parameters for any of the components, check the **Properties** tab in the HAL window for the respective driver (see Adding and Configuring HAL Drivers).

#### 3.3.4.5 Where is main()?

The main function is located in < project >/ra_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files, see Adding and Configuring HAL Drivers.

#### 3.3.4.6 Blinky Example Code

The blinky application is stored in the hal_entry.c file. This file is generated by e² studio when you select the Blinky Project template and is located in the project's src/ folder.

The application performs the following steps:

1. Get the LED information for the selected board by bsp_leds_t structure.
2. Define the output level HIGH for the GPIO pins controlling the LEDs for the selected board.
3. Get the selected system clock speed and scale down the clock, so the LED toggling can be observed.
4. Toggle the LED by writing to the GPIO pin with R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);

### 3.3.5 Build the Blinky Project

Highlight the new project in the **Project Explorer** window by clicking on it and build it.

There are three ways to build a project:

1. Click on **Project** in the menu bar and select **Build Project**.
2. Click on the hammer icon.
3. Right-click on the project and select **Build Project**.

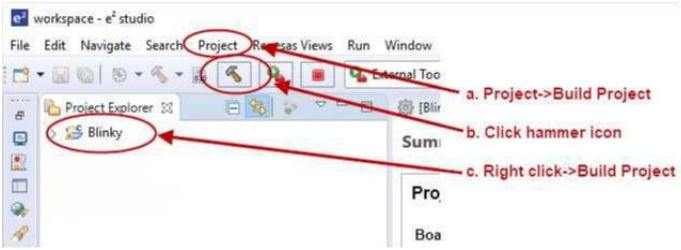



Figure 79: e² studio Project Explorer window

Once the build is complete a message is displayed in the build **Console** window that displays the final image file name and section sizes in that image.

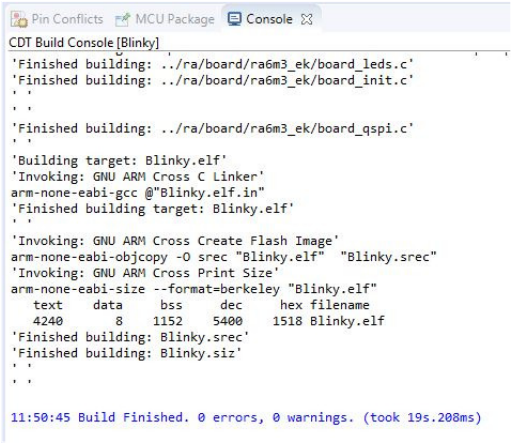



Figure 80: e² studio Project Build console

## 3.3.6 Debug the Blinky Project

### 3.3.6.1 Debug prerequisites

To debug the project on a board, you need

- The board to be connected to e² studio
- The debugger to be configured to talk to the board
- The application to be programmed to the microcontroller

Applications run from the internal flash of your microcontroller. To run or debug the application, the application must first be programmed to the microcontroller's flash. There are two ways to do this:

- JTAG debugger
- Built-in boot-loader via UART or USB

Some boards have an on-board JTAG debugger and others require an external JTAG debugger connected to a header on the board.

Refer to your board's user manual to learn how to connect the JTAG debugger to e² studio.

### 3.3.6.2 Debug steps

To debug the Blinky application, follow these steps:

1. Configure the debugger for your project by clicking **Run > Debugger Configurations ...**

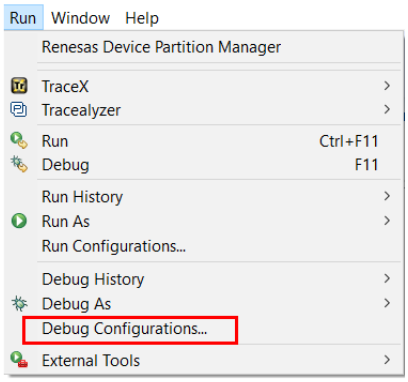



Figure 81: e² studio Debug icon

or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configurations ...**

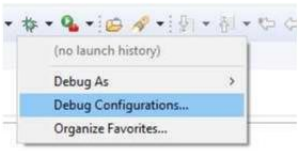



Figure 82: e² studio Debugger Configurations selection option

2. Select your debugger configuration in the window. If it is not visible then it must be created by clicking the **New** icon in the top left corner of the window. Once selected, the **Debug Configuration** window displays the Debug configuration for your Blinky project.

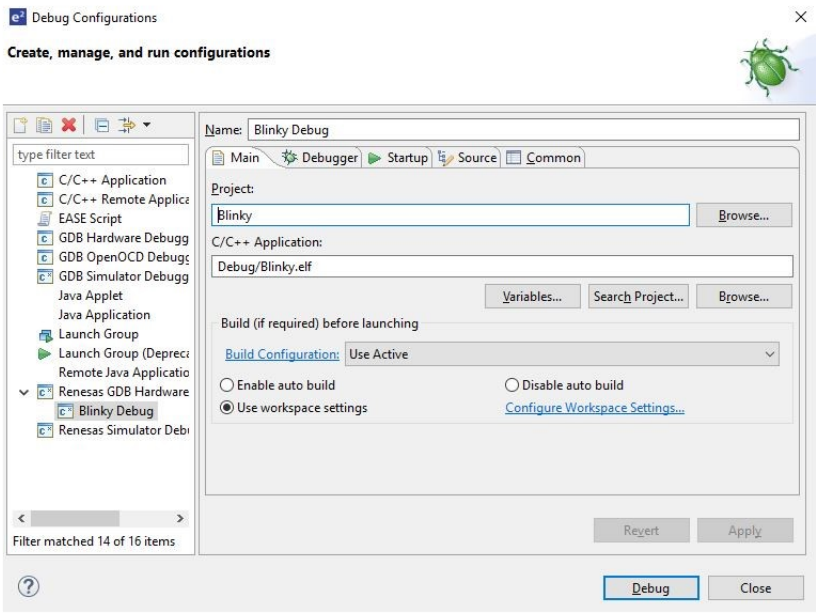
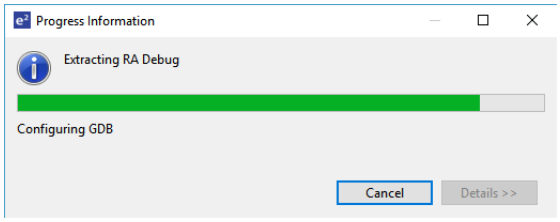



Figure 83: e² studio Debugger Configurations window with Blinky project

3. Click **Debug** to begin debugging the application.

4. Extracting RA Debug.




### 3.3.6.3 Details about the Debug Process

In debug mode, e² studio executes the following tasks:

1. Downloading the application image to the microcontroller and programming the image to the internal flash memory.
2. Setting a breakpoint at main().
3. Setting the stack pointer register to the stack.
4. Loading the program counter register with the address of the reset vector.
5. Displaying the startup code where the program counter points to.

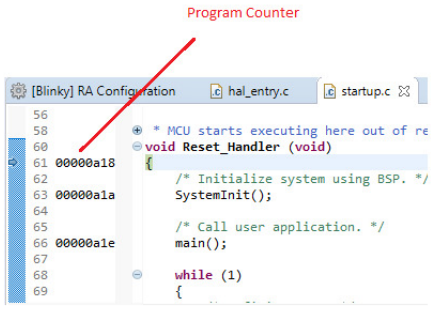



Figure 84: e² studio Debugger memory window

### 3.3.7 Run the Blinky Project

While in Debug mode, click **Run > Resume** or click on the **Play** icon twice.

Figure 85: e² studio Debugger Play icon

The LEDs on the board marked LED1, LED2, and LED3 should now be blinking.

## 3.4 Tutorial: Your First RA Multicore Application - Blinky

### 3.4.1 Introduction

This section provides information on how to create and debug multicore projects using e² studio and FSP. It is recommended that users read the section Tutorial: Your First RA MCU Project - Blinky before this one. The multicore application created will consist of a single project for each core of an RA8P1. Multicore support was added to FSP in v6.0.0 and the features available will be expanding in future releases. This section will be updated as needed.

**Note:** This section assumes the user already has a working knowledge of normal (non-multicore) usage of RA FSP and e² studio.

### 3.4.2 Generate Multicore Project Pair

e² studio now offers a mechanism to generate a multicore project pair using a single run through the project wizard. This is accessed via the "Renesas RA FSP Solution" option (rather than using the traditional C/C++ Project option):

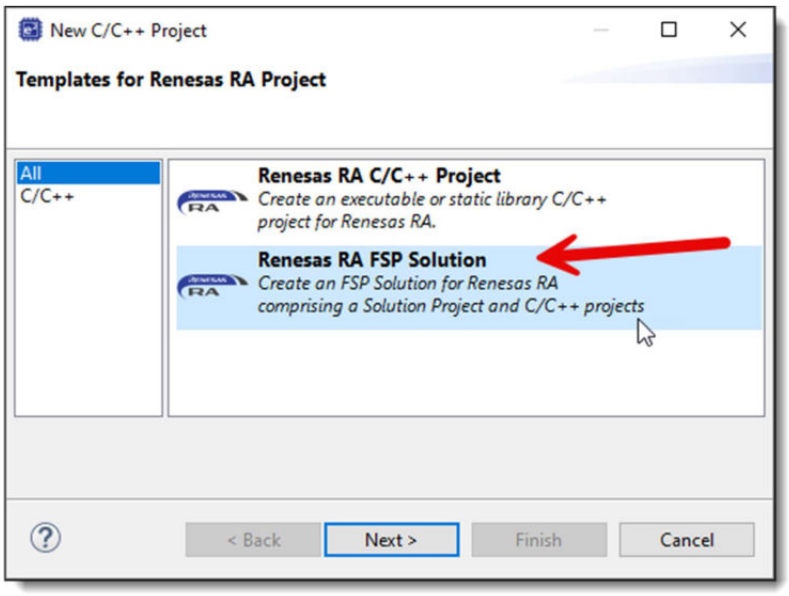



Figure 86: Project Generator - RA FSP Solution

When this option is selected, a similar set of dialog pages to the standard wizard will then be displayed, but currently reduced in number and with fewer options available.

Enter a name that will be used for the Solution project and the prefix for individual core projects. For example, if you choose a project name of "my_application" for an RA MCU with 2 cores, then there will be 3 resulting projects: "my_application" (Solution), "my_application_CPU0", and "my_application_CPU1".

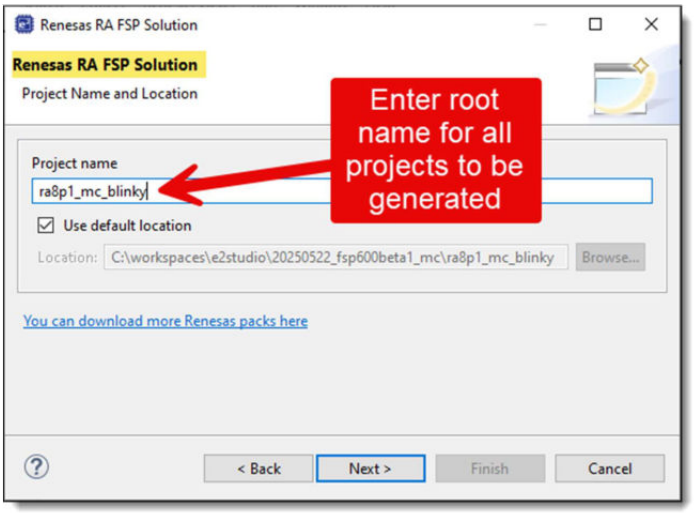



Figure 87: Project Generator - Enter Root Name

Choose the board you are using and confirm the toolchain you wish to use.

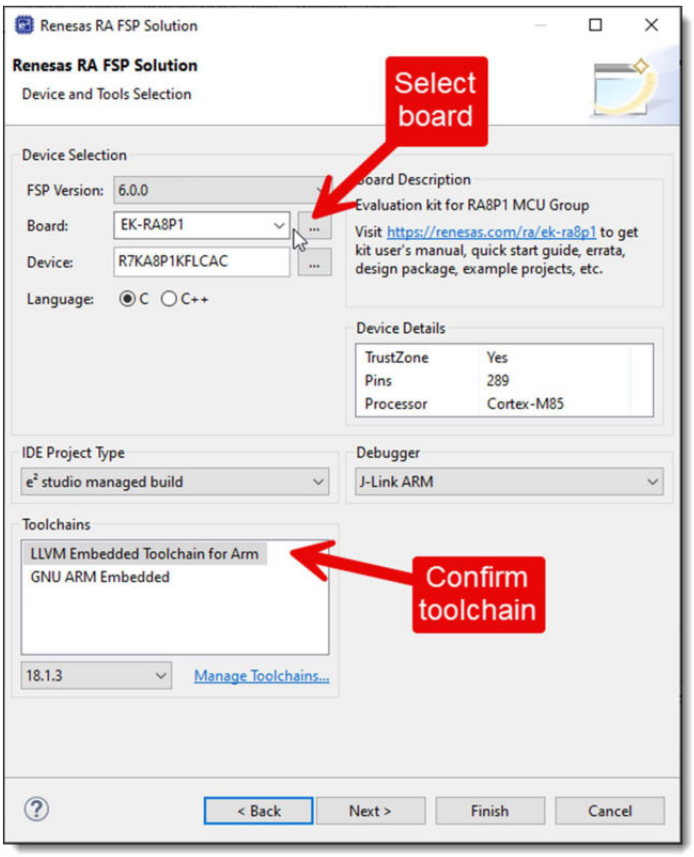



Figure 88: Project Generator - Configure Setup

Choose the "Bare Metal - Blinky" project template. This template will be used for creating the projects for all cores.

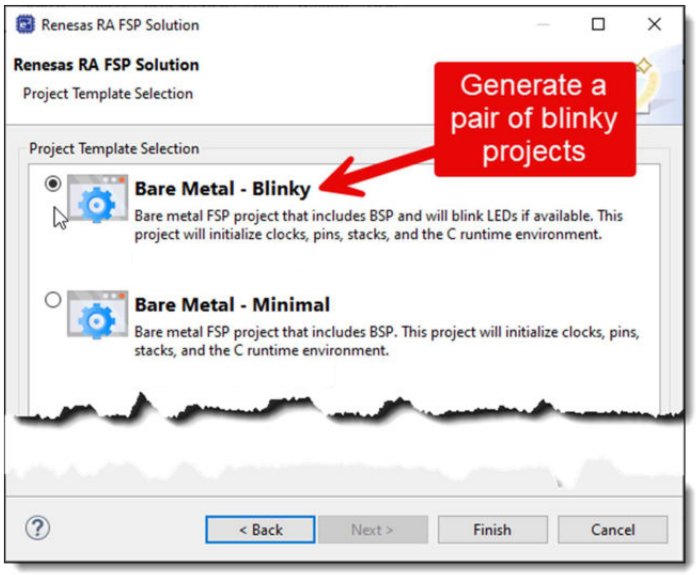



Figure 89: Project Generator - Choose Template

After clicking on the Finish button in the wizard, a set of projects will be generated with appropriate

linkage between them. Two of the projects are "normal" projects independent to each core which, when combined, will generate the overall multicore application that will be downloaded to the target MCU. The third project is the "container" Solution project which contains and controls configuration of the overall MCU and the CPU projects.

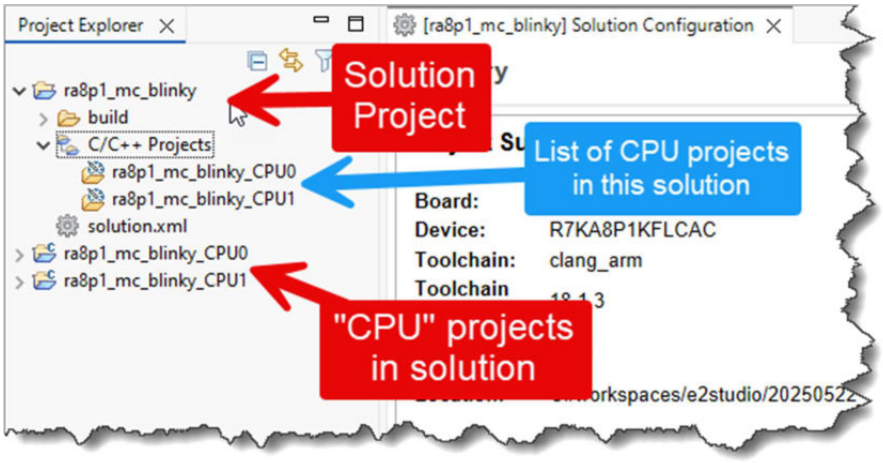



Figure 90: FSP Solution - Project Overview

*Note*

*After project creation, in order to carry out the population of some of the configuration data stored in the central solution project an initial build of both of the created CPU projects will be automatically performed.*

### 3.4.3 Information On Multicore Related Changes To hal_entry.c

Starting in FSP 6.0.0, the hal_entry.c file generated as part of the Blinky project template has been updated to support multicore projects. First, a call to the FSP inline function R_BSP_SecondaryCoreStart() (that starts execution of CPU1 by releasing it from reset) has been added to hal_entry() in the case that this file is being built for the primary core in a set of multicore projects.

Figure 91: Multicore Blinky - Starting secondary core

Second, the LED flashing code in hal_entry() has been changed so that it only blinks one of the LEDs for a multicore project (rather than all LEDs on the board).

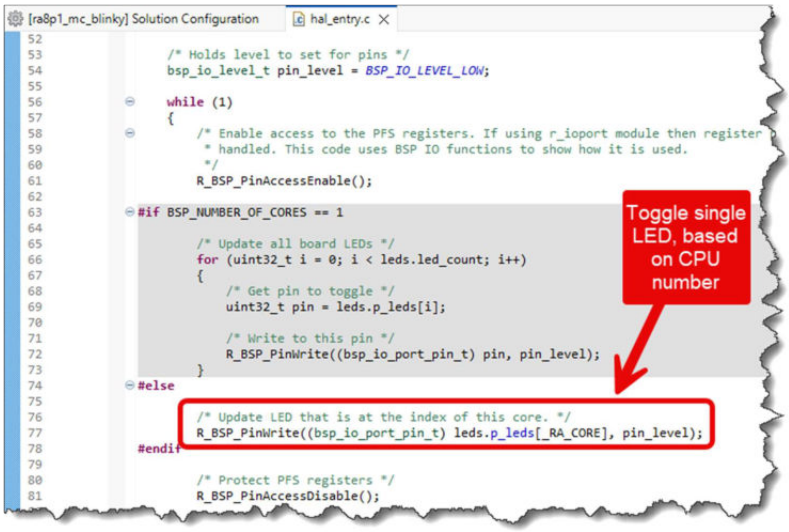



Figure 92: Multicore Blinky - Updated Code

This means that for a multicore application the primary core (CPU0) will blink the first LED on the board and the secondary core (CPU1) will blink the second LED. This allows the user to easily tell which core or cores are currently active.

## 3.4.4 Build Projects

When a FSP Solution application is used, builds should always be triggered from the Solution project. The reason is that the Solution project stores information that is applicable to all downstream projects. Building from the Solution ensures that all the latest settings are propagated to each downstream project.

*Note*

*To trigger a Solution build right-click on the Solution project and choose Build Project. In e² studio 2025-04.1 building Solutions by clicking the hammer icon is not available.*

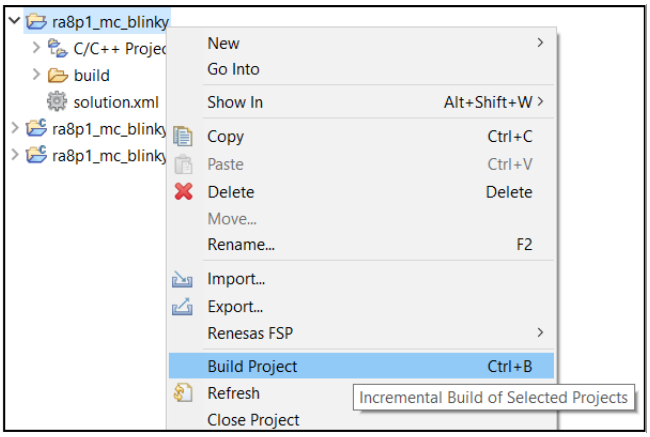



Figure 93: Solution Build

## 3.4.5 Information On Multicore Related Changes To Launch Configs

Although it is possible to start a debug session for CPU0 then manually start a second debug session for CPU1, it is easier to have e² studio do the two operations in one.

To enable this, the multicore solutions project wizard will create a "Launch Group" inside the CPU1 project containing the two individual launch configurations, one for each of the CPUs. This launch group is what will then be used to start a multicore debug session rather than the launch configurations themselves being used directly.

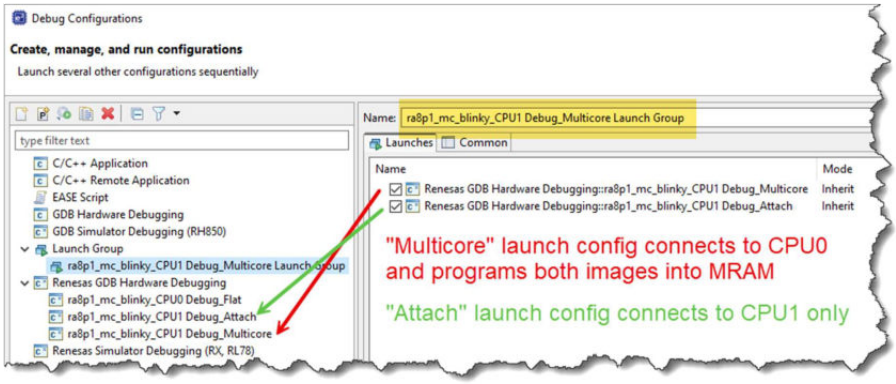



Figure 94: Multicore Debug - Launch Group

Note that the launch configuration generated inside the CPU0 project is generally not used when debugging the multicore application, as it simply connects to CPU0 and only programs the CPU0 image.

## 3.4.6 Reset TrustZone Partition Boundaries

**Important note: Currently the debug of multicore applications does not support the configuration of the TrustZone partition boundaries inside the MCU.** Therefore before carrying out debug of a multicore application for the first time, it is necessary to use either RFP or RDPM to do an initialize operation of your MCU to ensure that the whole Code ROM (Flash/MRAM) area is marked as Secure. RDPM can be accessed from within e² studio by going to Run >> Renesas Debug Tools >> Renesas Device Partition Manager **while a non-Solution project is selected**. If this is not done, the download of your images and debug startup may not work correctly. This will be improved in a future release of FSP and e² studio.

The image below shows the RDPM settings to initialize an EK-RA8P1. Note that "Initialize device" is selected and the Connection Type was changed to "SWD". It is recommended to power cycle the board after using RDPM and before debugging.

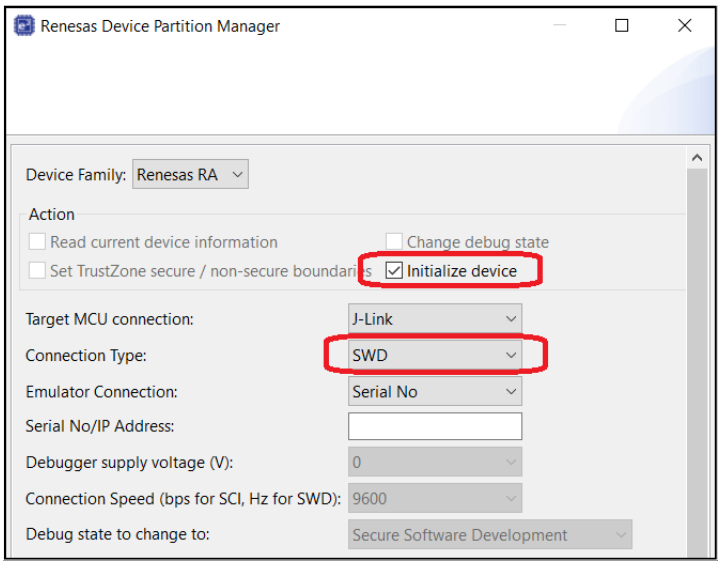



Figure 95: Multicore Debug - RDPM

### 3.4.7 Start Debugging

The first time you start debugging your multicore application, you will need to open the Debug Configurations dialog:

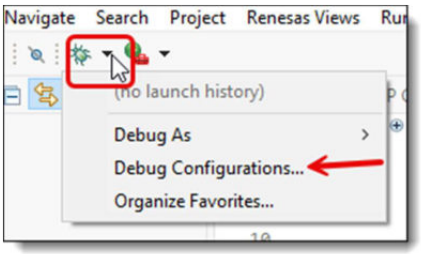



Figure 96: Multicore Debug - Debug Configuration

You will see the launch group that has been created. Select the Multicore Launch Group, then click the Debug button to start the debug session.

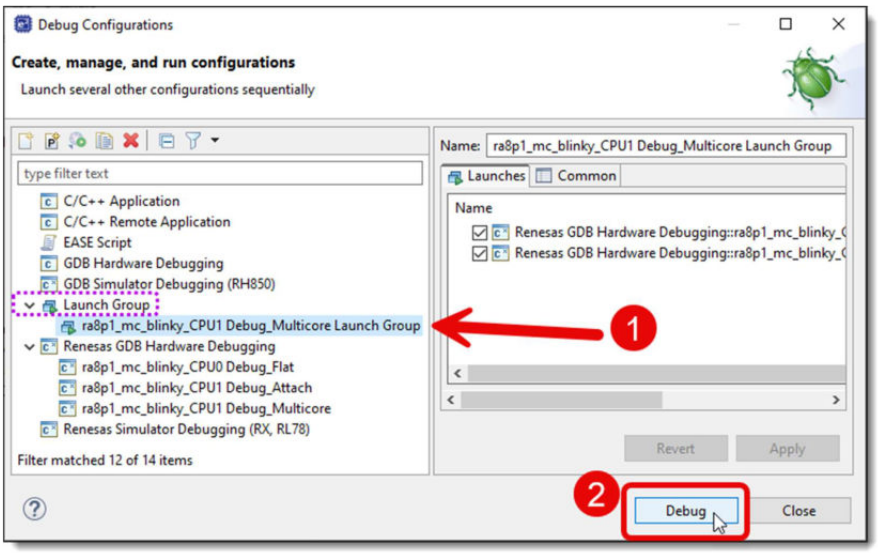



Figure 97: Multicore Debug - Start Launch Group

You should see the CPU0 project debug session start up and connect, followed by the CPU1 project. CPU1 will be shown in the Debug tree but will still be held in reset.

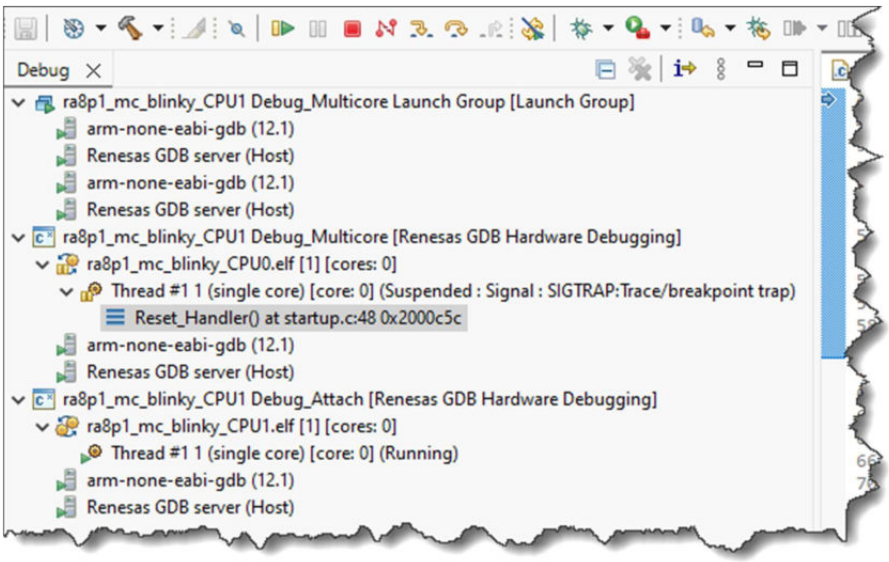



Figure 98: Multicore Debug - Reset

With the Reset_Handler() of the CPU0 elf file highlighted, if you now resume, CPU0 will start running. Once you pass main(), one of the LEDs on your board will begin to blink.

Once the call to R_BSP_SecondaryCoreStart() has been made from the CPU0 code, the CPU1 debug connection will also become fully active. At this point you can control CPU1 as you would CPU0.

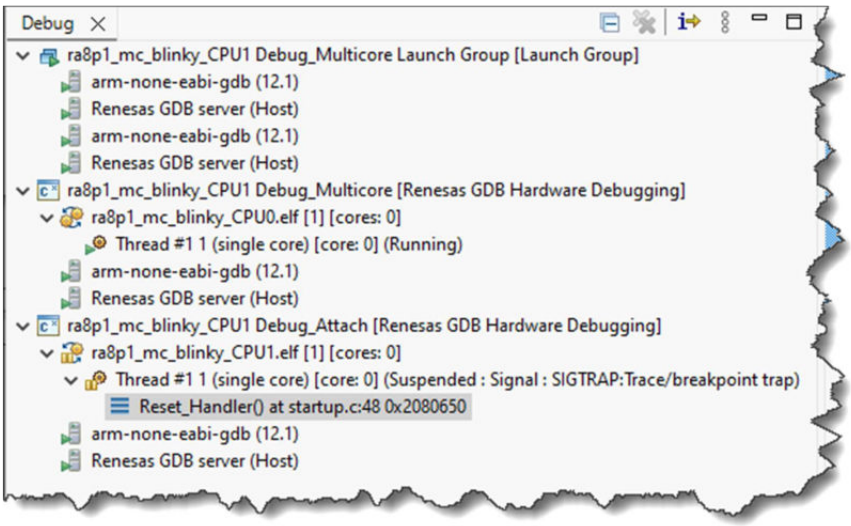



Figure 99: Multicore Debug - CPU1 Running

In the above screenshot, the Reset_Handler() of the CPU1 project is highlighted. Clicking Resume will start running code on CPU1. Once execution of main() on CPU1 has started, the 2nd LED on your board will also begin to blink.

## 3.4.8 Multicore Debugging Control

Once connected to both cores, the currently active debug connection (as highlighted in the Debug view) is the one that will be controlled by the usual debug controls (step, resume, suspend, etc).

However it is possible to have these buttons control both debug connections in parallel, by selecting both connections at once in the Debug View (using the control key to multi-select):

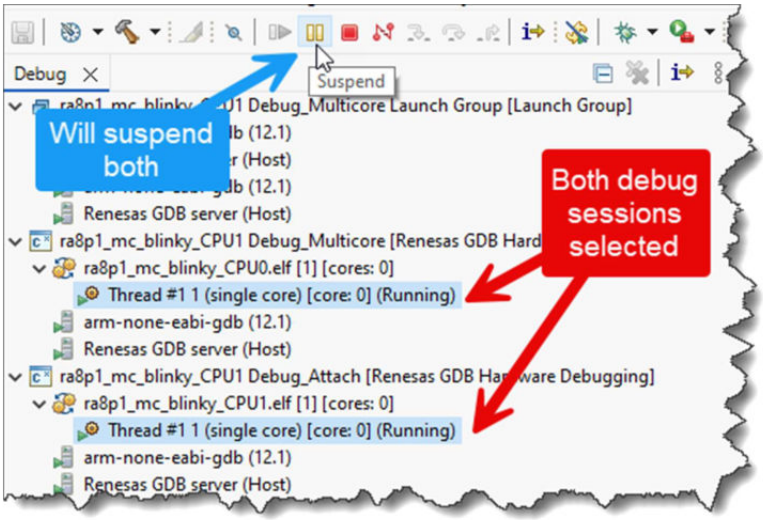



Figure 100: Multicore Debug - Suspend Both Cores

Note that the synchronization of operations here is being done at the software level within the GUI (cycling through each debug connection in turn, carrying out the requested operation). Although not

fully lockstepped at the MCU hardware level, the functionality this provides is still useful in many circumstances.

With regard to other views such as Registers, Memory, and Breakpoints, these will show the MCU state as seen by the currently selected core. However it is possible to pin each of these to a particular core, and also to clone the views so that you can have one pin to each core.

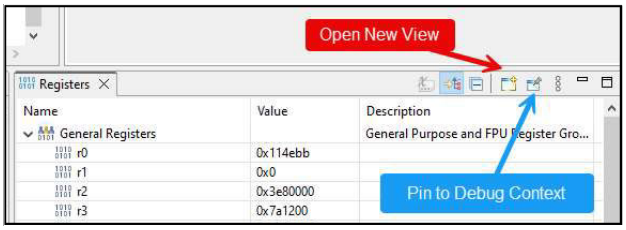



Figure 101: Multicore Debug - Pin View

## 3.4.9 Terminating a Multicore Debugging Session

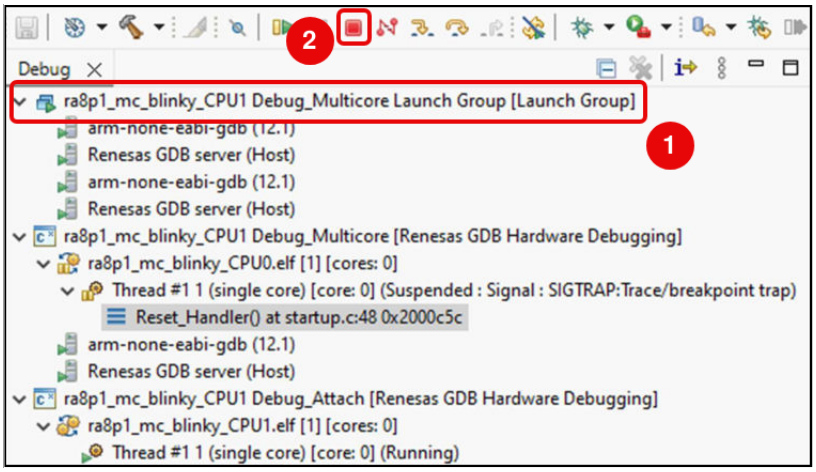



Figure 102: Multicore Debug - Terminate

## 3.4.10 Starting Future Debug Sessions

Once you have carried out your first multicore debug session, you can start further debug sessions simply using the appropriate Launch Group option which will then appear on the drop down menu:

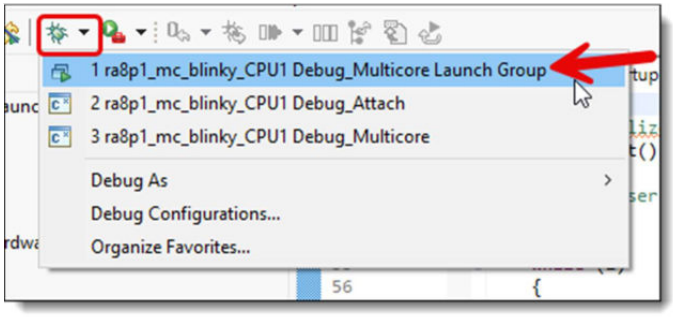



Figure 103: Multicore Debug - Starting Again

# 3.5 Tutorial: Using HAL Drivers - Programming the WDT

## 3.5.1 Application WDT

This tutorial illustrates the creation of a simple application that uses the Watchdog Timer module to monitor program operation. The tutorial shows each step in the development process and in particular identifies the auto-generated files and project structure created when using FSP and its GUI based configurator. The level of detail provided here is more than is normally needed during development but can be helpful in explaining how FSP works behind the scenes to simply your work.

This application makes use of the following FSP modules:

- MCU Board Support Package
- Watchdog (r_wdt)
- I/O Port (r_ioport)

## 3.5.2 Creating a WDT Application Using the RA MCU FSP and e² studio

### 3.5.2.1 Using FSP and e² studio

The Flexible Software Package (FSP) from Renesas provides a complete driver library for developing RA MCU applications. FSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers for the developer to use to create applications. FSP is integrated into Renesas e² studio based on eclipse providing build (editor, compiler and linker) and debug phases with an extended GNU Debug (GDB) interface.

### 3.5.2.2 The WDT Application

The flowchart for the WDT application is shown below.

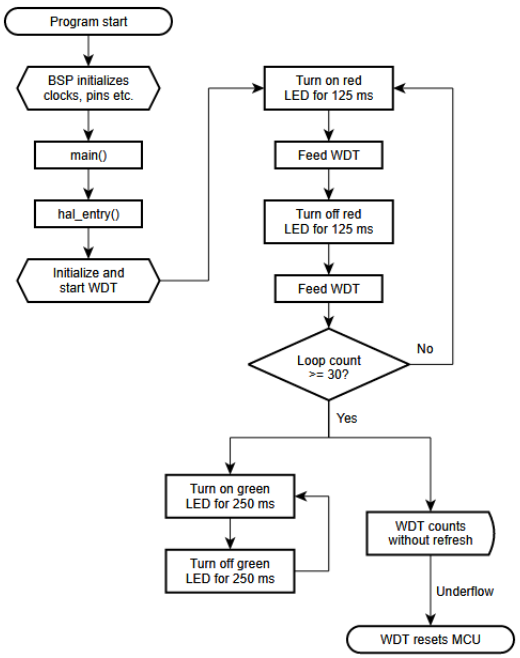



Figure 104: WDT Application flow diagram

#### 3.5.2.3 WDT Application flow

The main sections of the WDT application are:

1. The BSP initializes the clocks, pins and other elements of the MCU readying the application to run.
2. main() calls hal_entry(). The function hal_entry() is created by FSP with a placeholder for user code. The code for the WDT is added to this function.
3. Initialize the WDT, but do not start it.
4. Start the WDT by refreshing it.
5. In the first loop the red LED flashes 30 times and refreshes the watchdog each time the LED state is changed.
6. In the second loop, the green LED flashes, but the program DOES NOT refresh the watchdog. After the watchdog timeout period the device will reset which can be observed by the red LED flashing again as the sequence repeats.

### 3.5.3 Creating the Project with e² studio

Start e² studio and choose a workspace folder in the Workspace Launcher. Configure a new RA MCU project as follows.

1. Select **File > New > RA C/C++ Project**. Then select the template for the project.

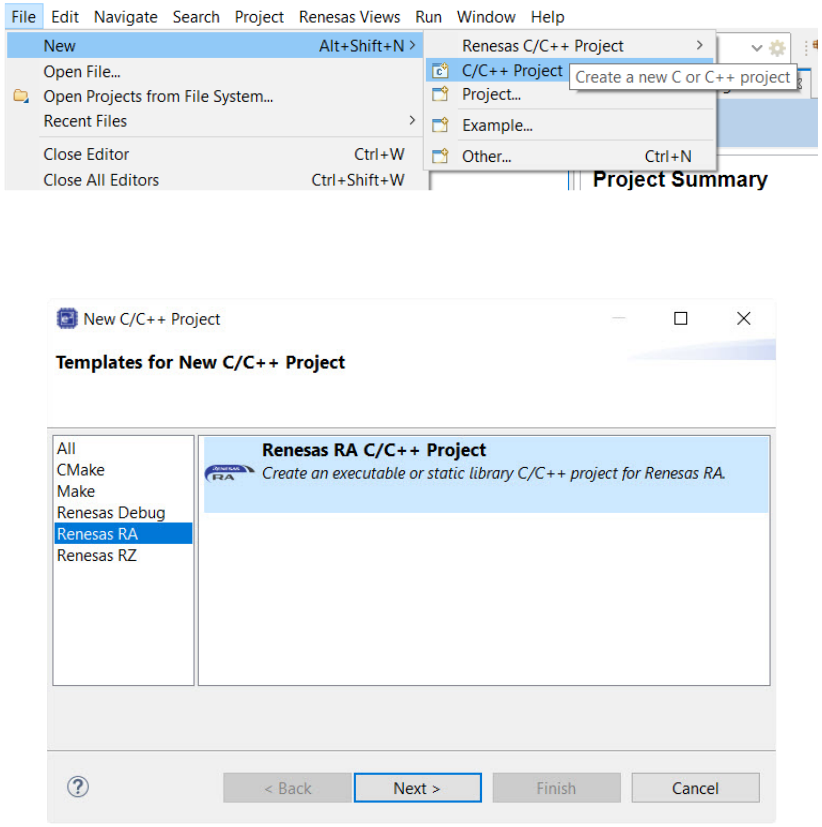



Figure 105: Creating a new project

2. In the e² studio Project **Configuration (RA Project)** window enter a project name, for example, WDT_Application. In addition, select the toolchain. If you want to choose new

locations for the project unselect **Use default location**. Click **Next**.

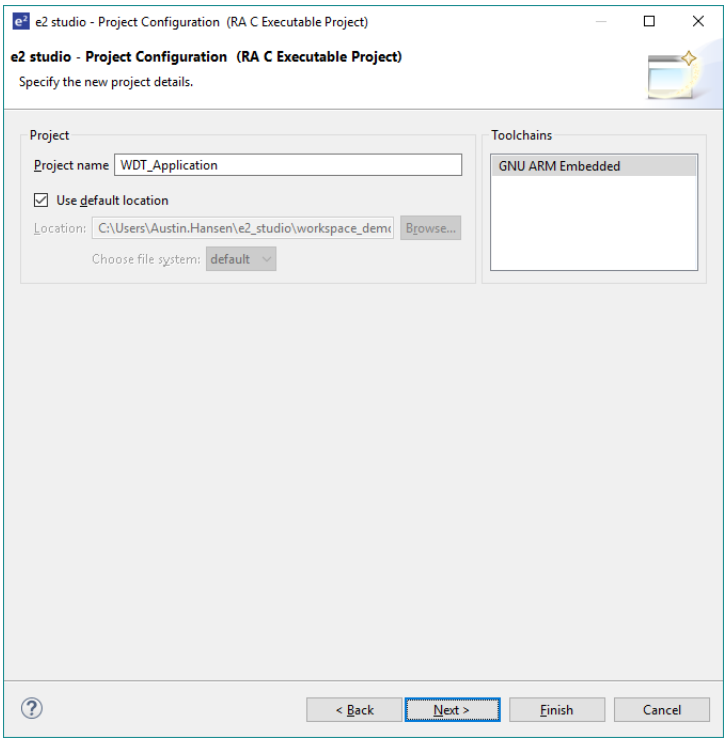



Figure 106: Project configuration (part 1)

3. This application runs on the EK-RA6M3 board. So, for the **Board** select **EK-RA6M3**.

   This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **J-Link ARM** as the **Debugger**. Click **Next** to configure the project.

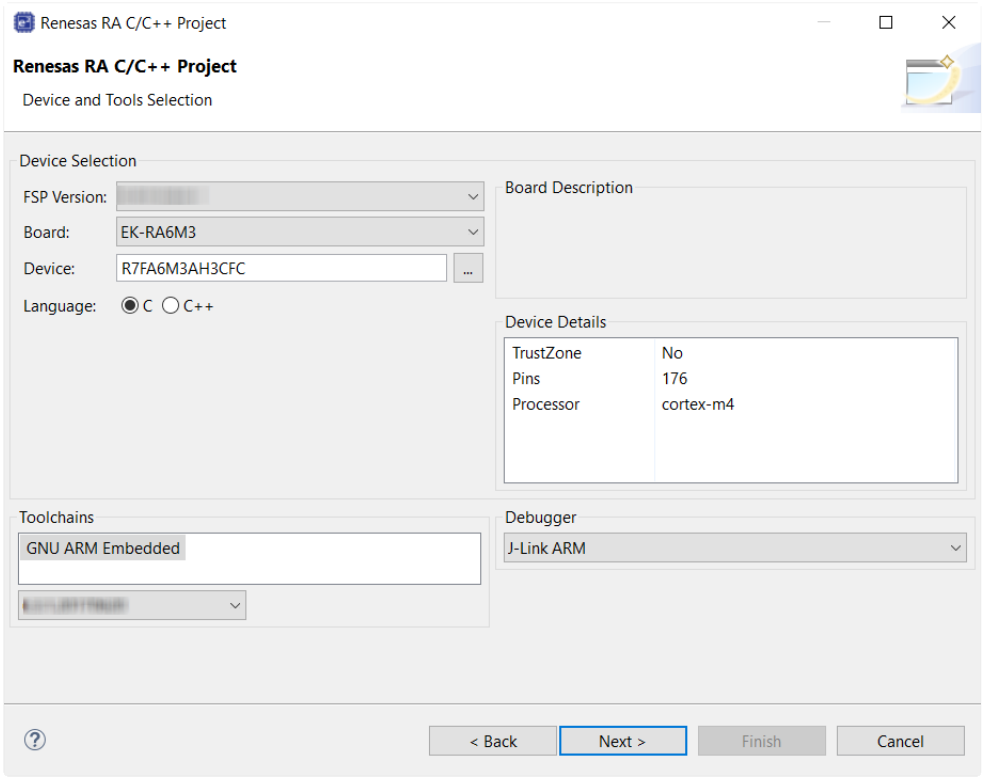



Figure 107: Project configuration (part 2)

The project template is now selected. As no RTOS is required select **Bare Metal - Blinky**.

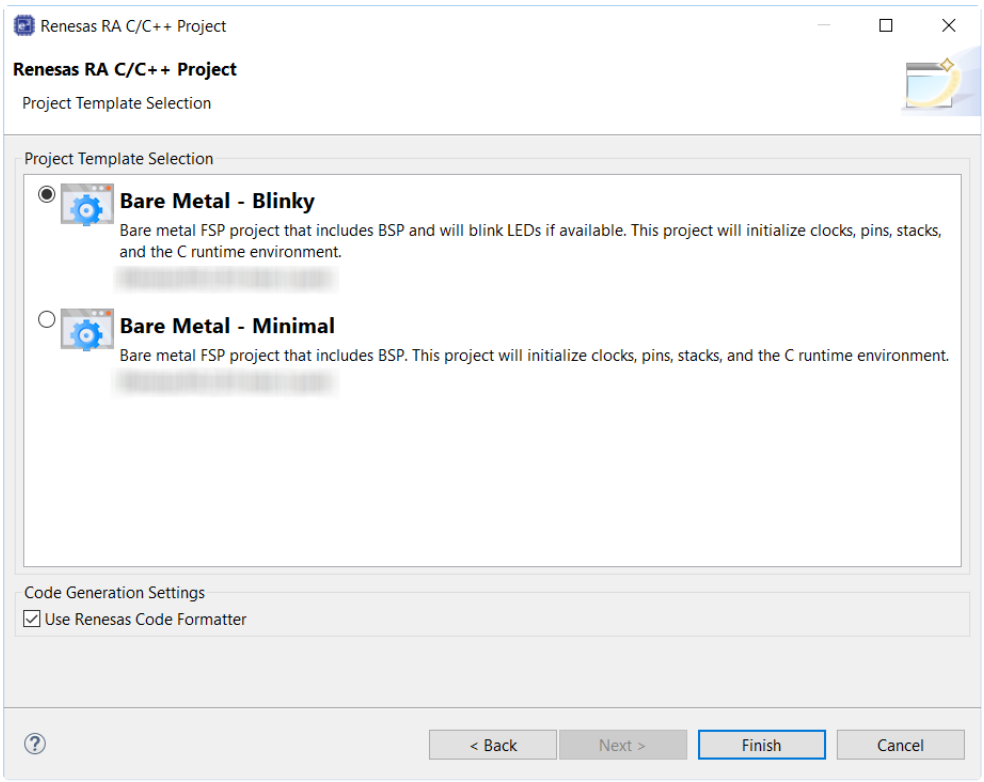



Figure 108: Project configuration (part 3)

4. Click **Finish**.

   e² studio creates the project and opens the **Project Explorer** and **Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

## 3.5.4 Configuring the Project with e² studio

e² studio simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

e² studio offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++**, **FSP Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right.

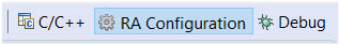



Figure 109: Selecting a perspective

The **C/C++** perspective provides a layout selected for code editing. The **FSP Configuration** perspective provides elements for configuring a RA MCU project, and the **Debug** perspective provides a view suited for debugging.

1. In order to configure the project settings ensure the **FSP Configuration** perspective is selected.
2. Ensure the **Project Configuration [WDT Application]** is open. It is already open if the Summary information is visible. To open the Project Configuration now or at any time make sure the **RA Configuration** perspective is selected and double-click on the configuration.xml file in the Project Explorer pane on the right side of e² studio.

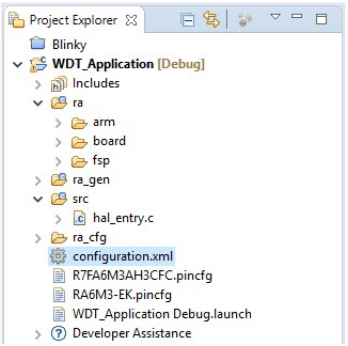



Figure 110: RA MCU Project Configuration Settings

At the base of the Project Configuration view there are several tabs for configuring the project. A project may require changes to some or all of these tabs. The tabs are shown below.

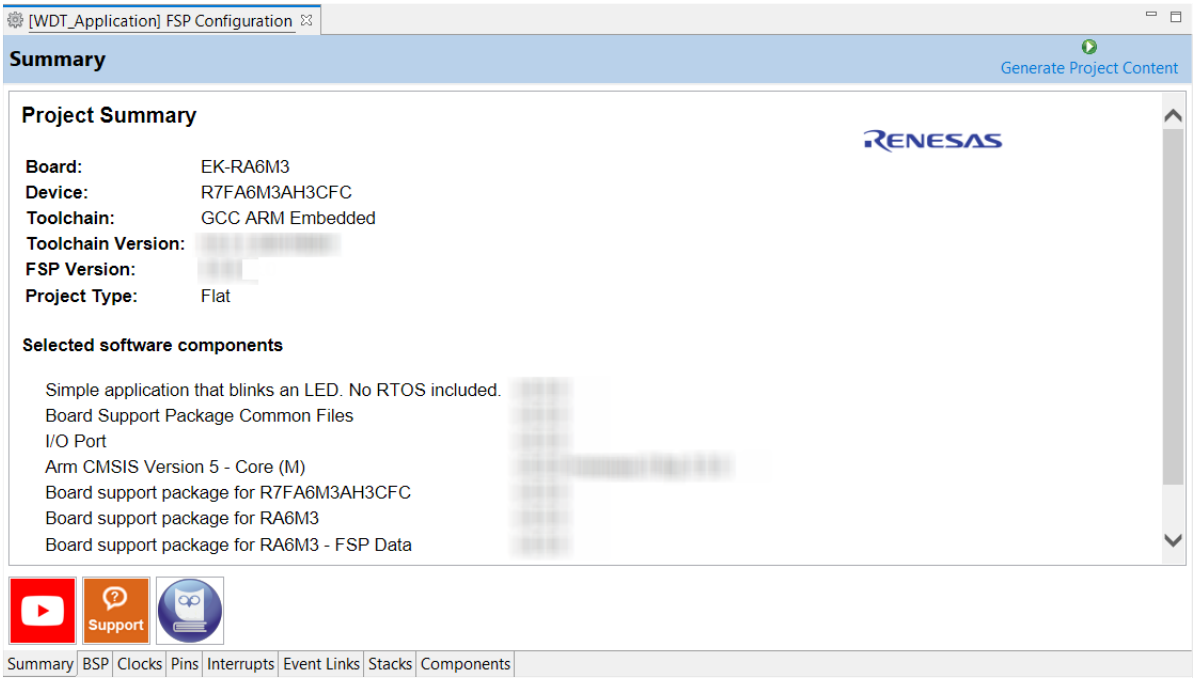



Figure 111: Project Configuration Tabs

### 3.5.4.1 BSP Tab

The **BSP** tab allows the Board Support Package (BSP) options to be modified from their defaults. For this particular WDT project no changes are required. However, if you want to use the WDT in auto-start mode, you can configure the settings of the OFS0 (Option Function Select Register 0) register in the **BSP** tab. See the RA Hardware User's Manual for details on the WDT autostart mode.

### 3.5.4.2 Clocks Tab

The **Clocks** tab presents a graphical view of the clock tree of the device. The drop-down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLCKB. The default output frequency for this clock is 60 MHz. Ensure this clock is outputting this value.

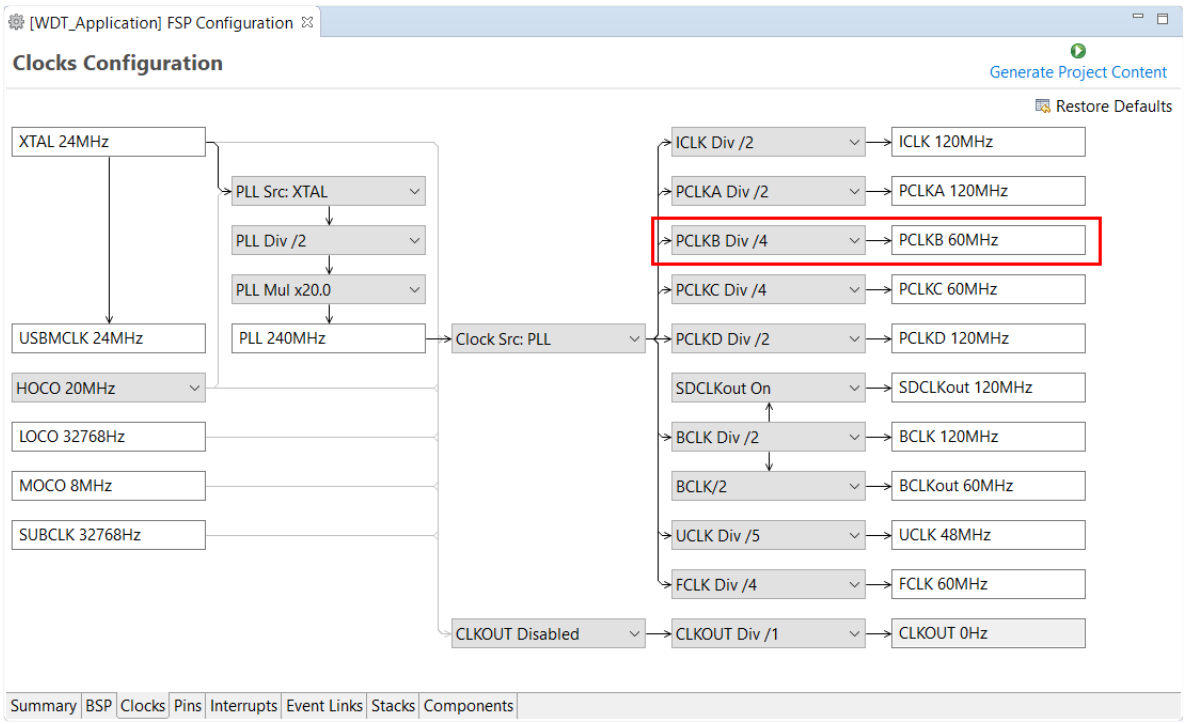



Figure 112: Clock configuration

### 3.5.4.3 Interrupts Tab

The **Interrupts** tab is used to add new user events or interrupts. No new interrupts or events are needed by the application, so no edits in this tab are required.

### 3.5.4.4 Event Links Tab

The **Event Links** tab is used to configure events used by the Event Link Controller (ELC). This project doesn't use the ELC, so no edits in this tab are required.

### 3.5.4.5 Pins Tab

The **Pins** tab provides a graphical tool for configuring the functionality of the pins of the device. For the WDT project no pin configuration is required. Although the project uses two LEDs connected to pins on the device, these pins are pre-configured as output GPIO pins by the BSP.

### 3.5.4.6 Stacks Tab

You can add any driver to the project using the **Stacks** tab. The HAL driver IO port pins are added automatically by e² studio when the project is configured. The WDT application uses no RTOS Resources, so you only need to add the HAL WDT driver.

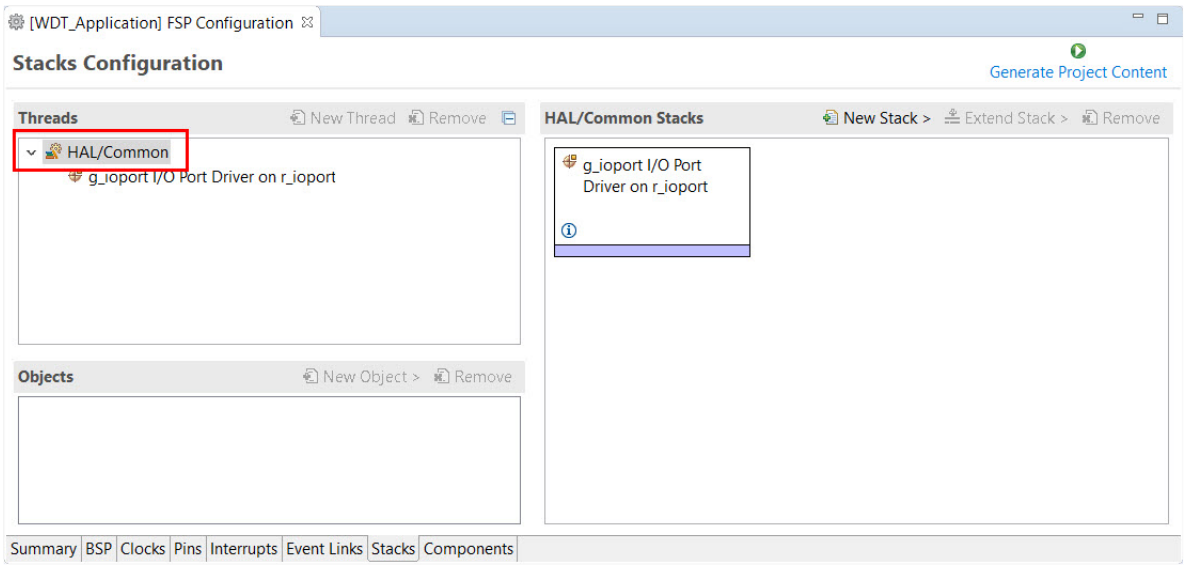



Figure 113: Stacks tab

1. Click on the **HAL/Common Panel** in the Threads Window as indicated in the figure above.

   The Stacks Panel becomes a **HAL/Common Stacks** panel and is populated with the modules preselected by e² studio.

2. Click on **New Stack** to find a pop-up window with the available HAL level drivers.
3. Select **WATCHDOG Driver on r_wdt**.

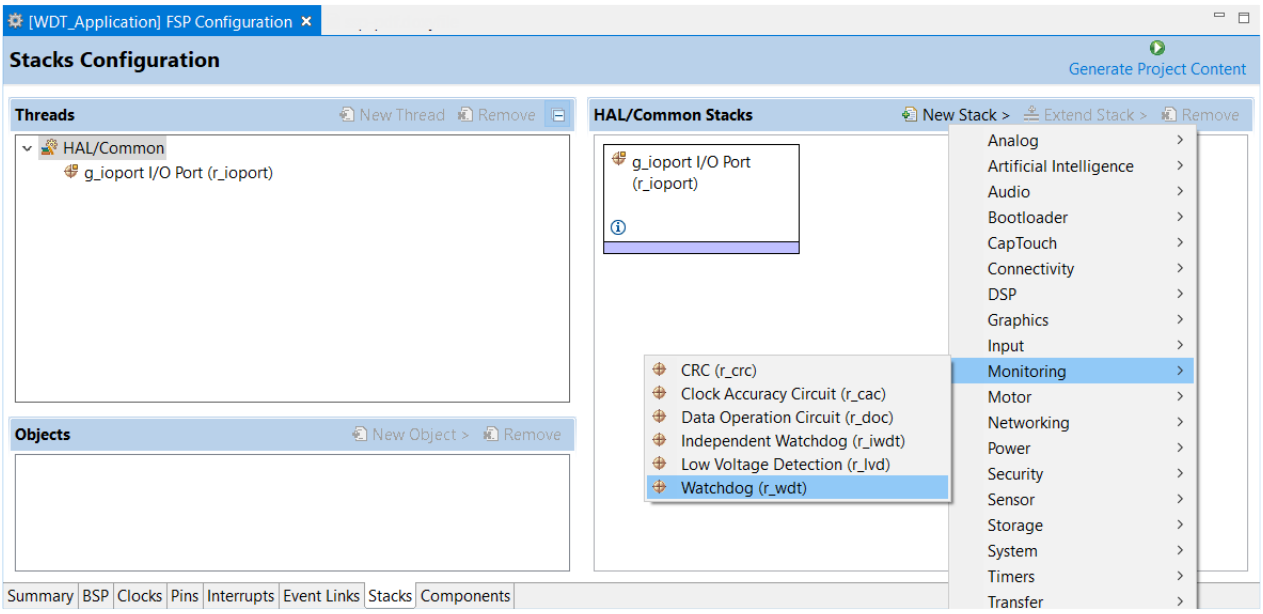



Figure 114: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Stacks** Panel and the **Property** Window shows all configuration options for the selected module. The **Property** tab for the WDT should be visible at the bottom left of the screen. If it is not visible, check that the **FSP Configuration** perspective is selected.

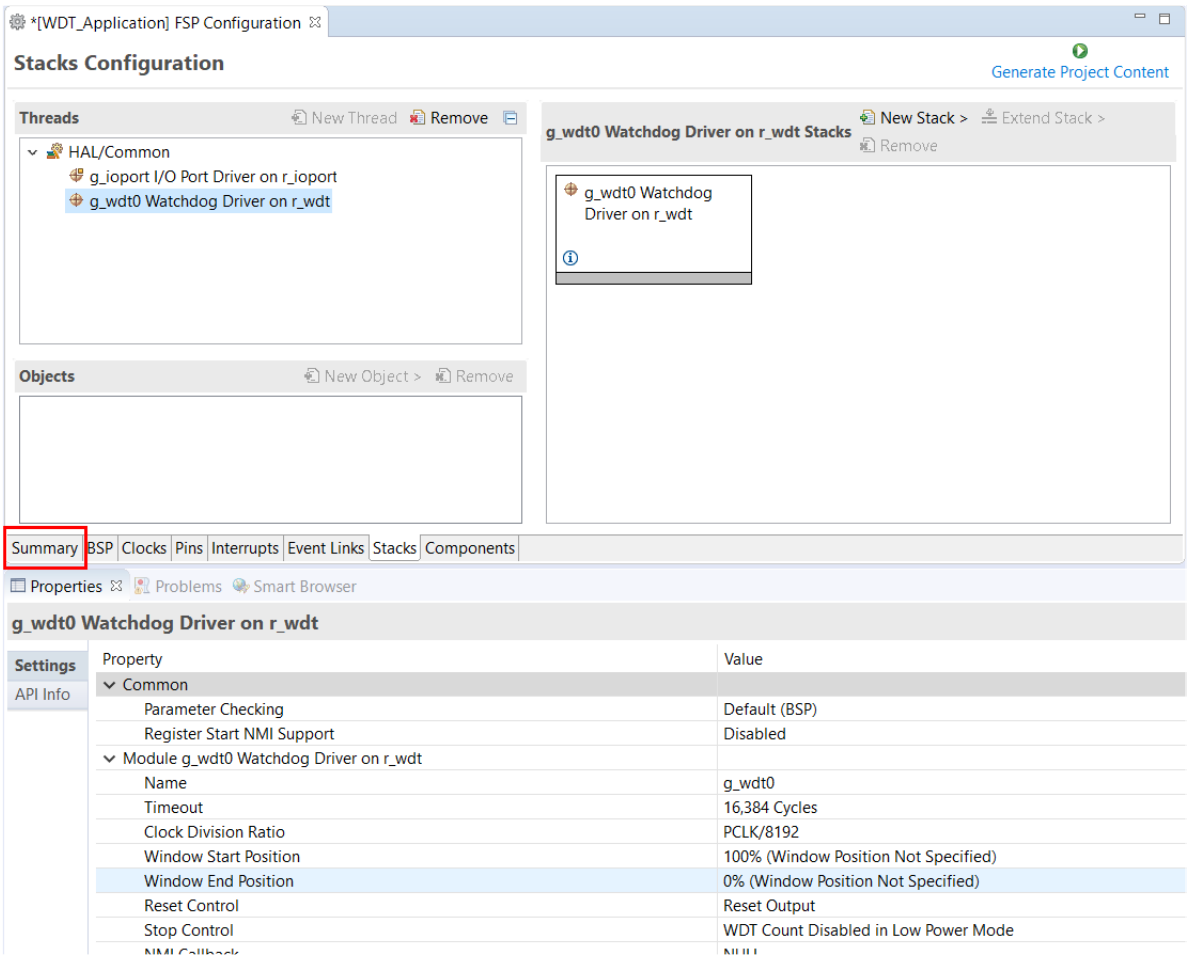



Figure 115: Module Properties

All parameters can be left with their default values.

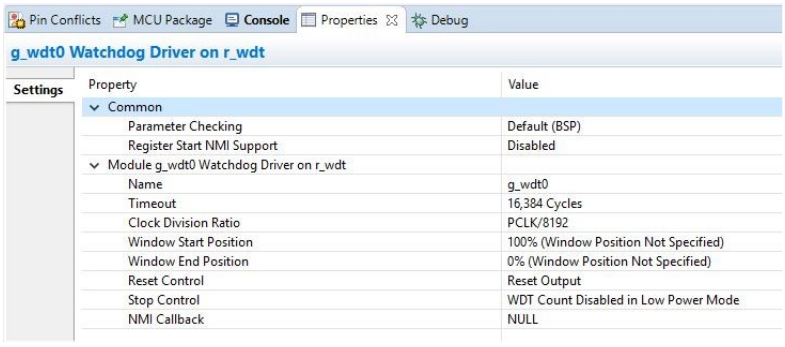



Figure 116: g_wdt WATCHDOG Driver on WDT properties

With PCLKB running at 60 MHz the WDT will reset the device 2.23 seconds after the last refresh.

WDT clock = 60 MHz / 8192 = 7.32 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 = 2.23 seconds

Save the **Project Configuration** file and click the **Generate Project Content** button in the top right corner of the **Project Configuration** pane.

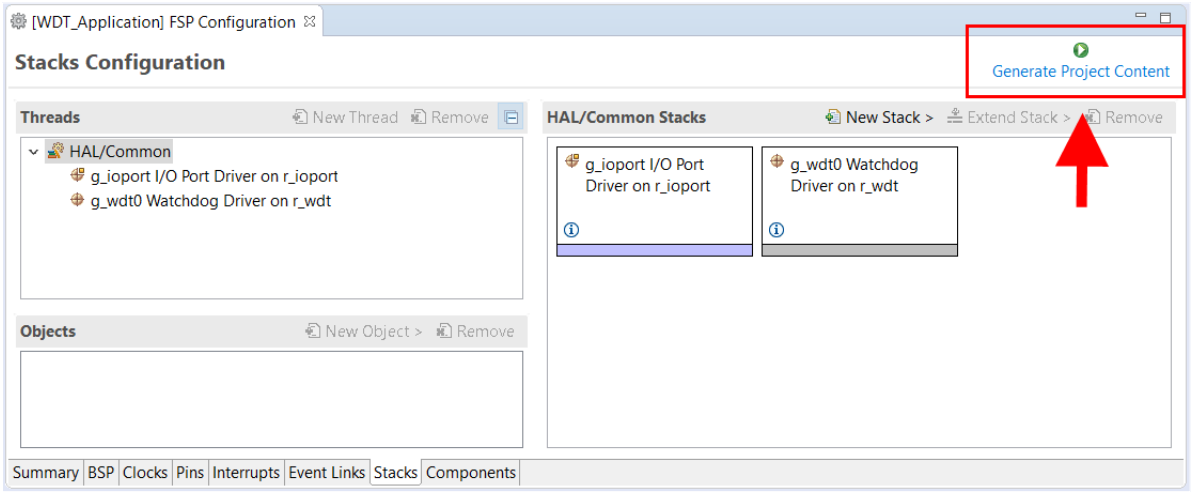



Figure 117: Generate Project Content button

e² studio generates the project files.

### 3.5.4.7 Components Tab

The components tab is included for reference to see which modules are included in the project. Modules are selected automatically in the Components view after they are added in the Stacks Tab.

For the WDT project ensure that the following modules are selected:

1. HAL_Drivers -> r_ioport
2. HAL_Drivers -> r_wdt

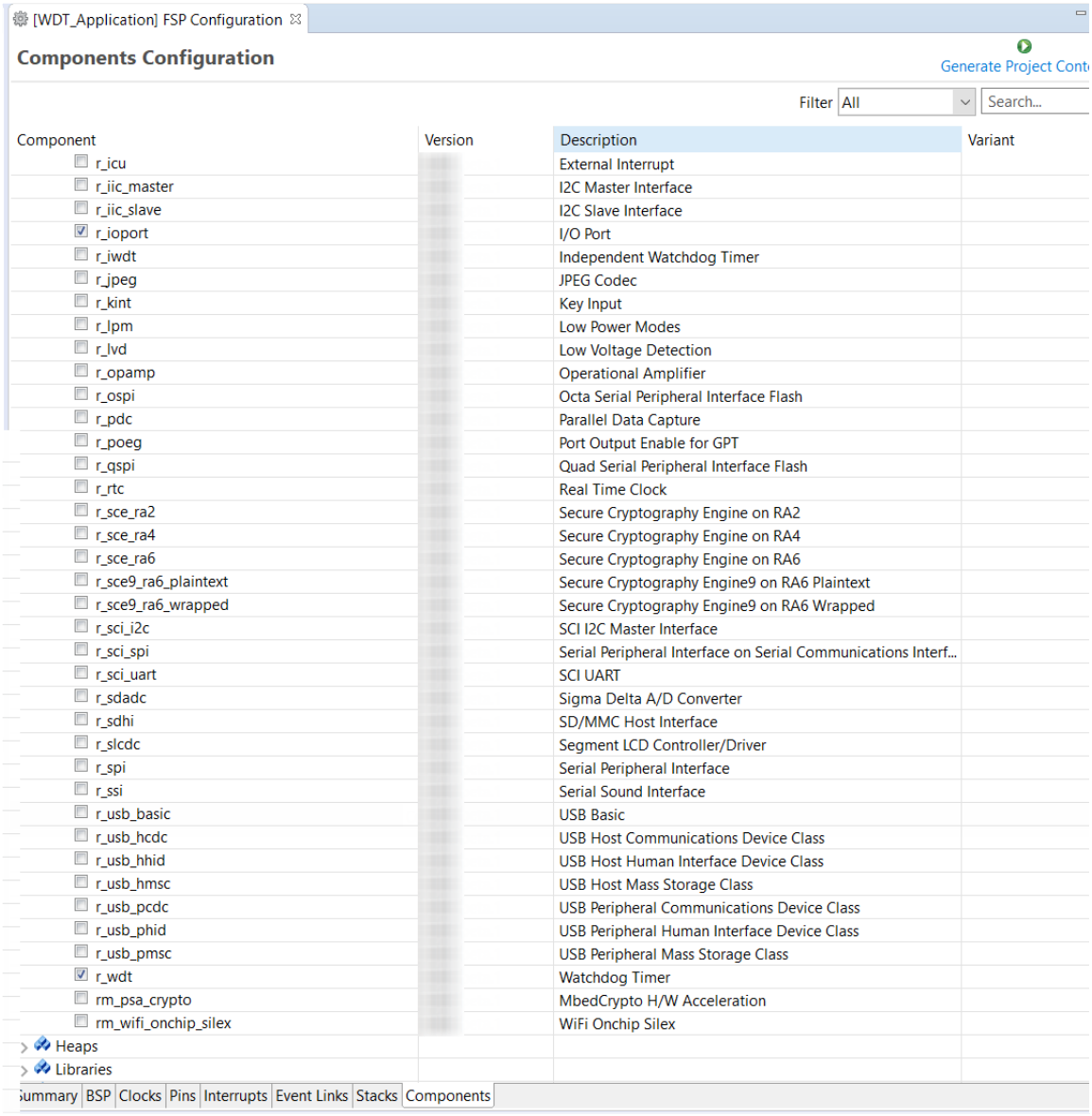



Figure 118: Component Selection

*Note*

*The list of modules displayed in the Components tab depends on the installed FSP version.*

## 3.5.5 WDT Generated Project Files

Clicking the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:

  ra/fsp/src

- r_wdt_api.h created in:

  ra/fsp/inc/api

- r_wdt.h created in:

  ra/fsp/inc/instances

The above files are the standard files for the WDT HAL module. They contain no specific project contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

Configuration information for the WDT HAL module in the WDT project is found in:

ra_cfg/fsp_cfg/r_wdt_cfg.h

The above file's contents are based upon the **Common** settings in the **g_wdt WATCHDOG Driver on WDT Properties** pane.

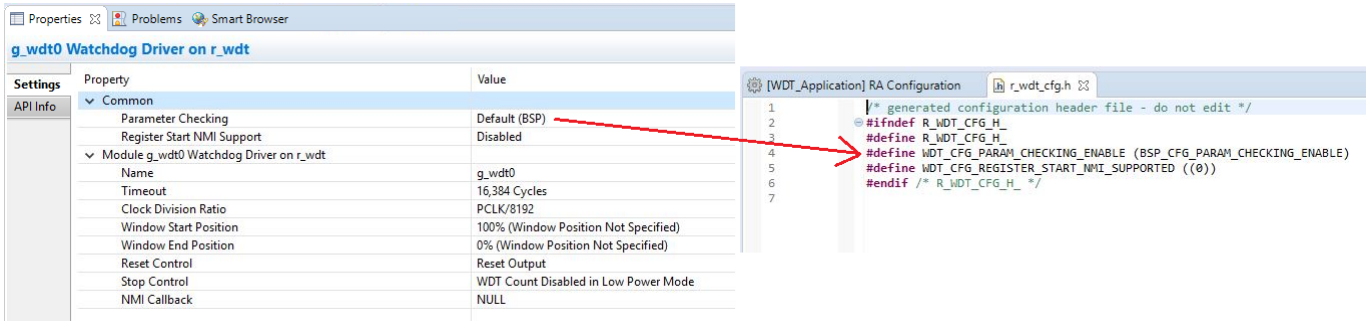



Figure 119: r_wdt_cfg.h contents

Warning

Do not edit any of these files as they are recreated every time the Generate Project Content button is clicked and so any changes will be overwritten.

The r_ioport folder is not created at ra/fsp/src as this module is required by the BSP and so already exists. It is included in the WDT project in order to include the correct header file in ra_gen/hal_data.c–see later in this document for further details. For the same reason the other IOPORT header files– ra/fsp/inc/api/r_ioport_api.handra/fsp/inc/instances/r_ioport.h–are not created as they already exist.

In addition to generating the HAL driver files for the WDT and IOPORT files e² studio also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

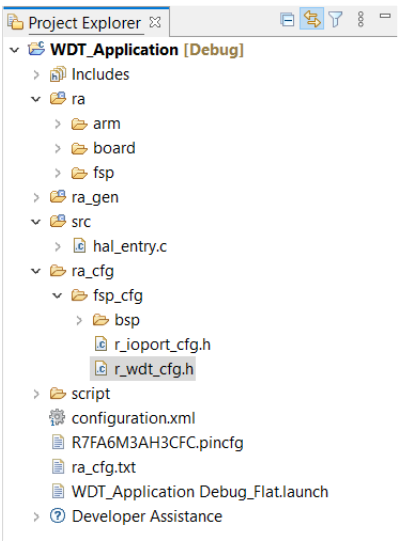



Figure 120: WDT project files

### 3.5.5.1 WDT hal_data.h

The contents of hal_data.h are shown below.

```
/* generated HAL header file - do not edit */
#ifndef HAL_DATA_H_
 #define HAL_DATA_H_
 #include <stdint.h>
 #include "bsp_api.h"
 #include "common_data.h"
 #include "r_wdt.h"
 #include "r_wdt_api.h"
 #ifdef __cplusplus
extern "C"
{
 #endif
extern const wdt_instance_t g_wdt0;
 #ifndef NULL
void NULL(wdt_callback_args_t * p_args);
 #endif
extern wdt_instance_ctrl_t g_wdt0_ctrl;
extern const wdt_cfg_t g_wdt0_cfg;
void hal_entry(void);
```

```
void g_hal_init(void);
 #ifdef __cplusplus
} /* extern "C" */
 #endif
#endif  /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the generated project. In addition this file includes external references to the **g_wdt0** instance structure which contains pointers to the configuration, control, api structures used for WDT HAL driver.

Warning

This file is regenerated each time Generate Project Content is clicked and must not be edited.

### 3.5.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */
#include "hal_data.h"
wdt_instance_ctrl_t g_wdt0_ctrl;
const wdt_cfg_t g_wdt0_cfg =
{
    .timeout       = WDT_TIMEOUT_16384,
    .clock_division = WDT_CLOCK_DIVISION_8192,
    .window_start   = WDT_WINDOW_START_100,
    .window_end    = WDT_WINDOW_END_0,
    .reset_control = WDT_RESET_CONTROL_RESET,
    .stop_control   = WDT_STOP_CONTROL_ENABLE,
    .p_callback    = NULL,
};
/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{.p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt};
void g_hal_init (void)
{
    g_common_init();
}
```

hal_data.c contains g_wdt0_ctrl which is the control structure for this instance of the WDT HAL driver. This structure should not be initialized as this is done by the driver when it is opened.

The contents of g_wdt0_cfg are populated in this file using the **Watchdog Driver on g_wdt0** pane in the Project Configuration **Stacks** tab. If the contents of this structure do not reflect the settings made in the IDE, ensure the **Project Configuration** settings are saved before clicking the **Generate Project Content** button.

Warning

This file is regenerated each time Generate Project Content is clicked and so should not be edited.

#### 3.5.5.3 WDT main.c

Contains main() called by the BSP start-up code. main() calls hal_entry() which contains user developed code (see next file). Here are the contents of main.c.

```
/* generated main source file - do not edit*/
#include "hal_data.h"
int main (void)
{
    hal_entry();
 return 0;
}
```

Warning

This file is regenerated each time Generate Project Content is clicked and so should not be edited.

#### 3.5.5.4 WDT hal_entry.c

This file contains the function hal_entry() called from main(). User developed code should be placed in this file and function.

For the WDT project edit the contents of this file to contain the code below. This code implements the flowchart in overview section of this document.

```
#include "hal_data.h"
#include "bsp_pin_cfg.h"
#include "r_ioport.h"
#define RED_LED_NO_OF_FLASHES 30
#define RED_LED_PIN BSP_IO_PORT_01_PIN_00
```

```
#define GREEN_LED_PIN BSP_IO_PORT_04_PIN_00
#define RED_LED_DELAY_MS 125
#define GREEN_LED_DELAY_MS 250
volatile uint32_t delay_counter;
volatile uint16_t loop_counter;
void R_BSP_WarmStart(bsp_warm_start_event_t event);
/*******************************************************************************
*******************************/
void hal_entry (void)
{
 /* Allow the WDT to run when the debugger is connected */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
 /* Open the WDT */
 R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Start the WDT by refreshing it */
 R_WDT_Refresh(&g_wdt0_ctrl);
 /* Flash the red LED and feed the WDT for a few seconds */
 for (loop_counter = 0; loop_counter < RED_LED_NO_OF_FLASHES; loop_counter++)
    {
 /* Turn red LED on */
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_HIGH);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
    }
 /* Flash green LED but STOP feeding the WDT. WDT should reset the
  * device */
 while (1)
```

```
    {
 /* Turn green LED on */
 R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(GREEN_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Turn green off */
 R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_HIGH);
 /* Delay */
 R_BSP_SoftwareDelay(GREEN_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
/*******************************************************************************
*******************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
 if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0
 /* Enable reading from data flash. */
       R_FACI_LP->DFLCTL = 1U;
 /* Would normally have to wait for tDSTOP(6us) for data flash recovery. Placing the
enable here, before clock and
  * C runtime initialization, should negate the need for a delay since the
initialization will typically take more than 6us. */
#endif
    }
 if (BSP_WARM_START_POST_C == event)
    {
 /* C runtime environment and system clocks are setup. */
 /* Configure pins. */
 R_IOPORT_Open(&IOPORT_CFG_CTRL, &IOPORT_CFG_NAME);
    }
}
```

The WDT HAL driver API functions are defined in r_wdt.h. The WDT HAL driver is opened through the open API call using the instance structure defined in r_wdt_api.h:

```
/* Open the WDT */
R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

The first passed parameter is the pointer to the control structure g_wdt0_ctrl instantiated in hal_data.c. The second parameter is the pointer to the configuration data g_wdto_cfg instantiated in the same hal_data.c file.

The WDT is started and refreshed through the API call:

```
/* Start the WDT by refreshing it */
R_WDT_Refresh(&g_wdt0_ctrl);
```

Again the first (and only in this case) parameter passed to this API is the pointer to the control structure of this instance of the driver.

### 3.5.6 Building and Testing the Project

Build the project in e² studio by clicking **Build > Build Project** or by clicking the build icon. The project should build without errors.

To debug the project

1. Connect the USB cable between the target board debug port and host PC.
2. In the **Project Explorer** pane on the left side of e² studio, right-click on the WDT project **WDT_Application** and select **Debug As > Debug Configurations**.

3. Under **Renesas GDB Hardware Debugging** select **WDT_Application Debug** as shown below.

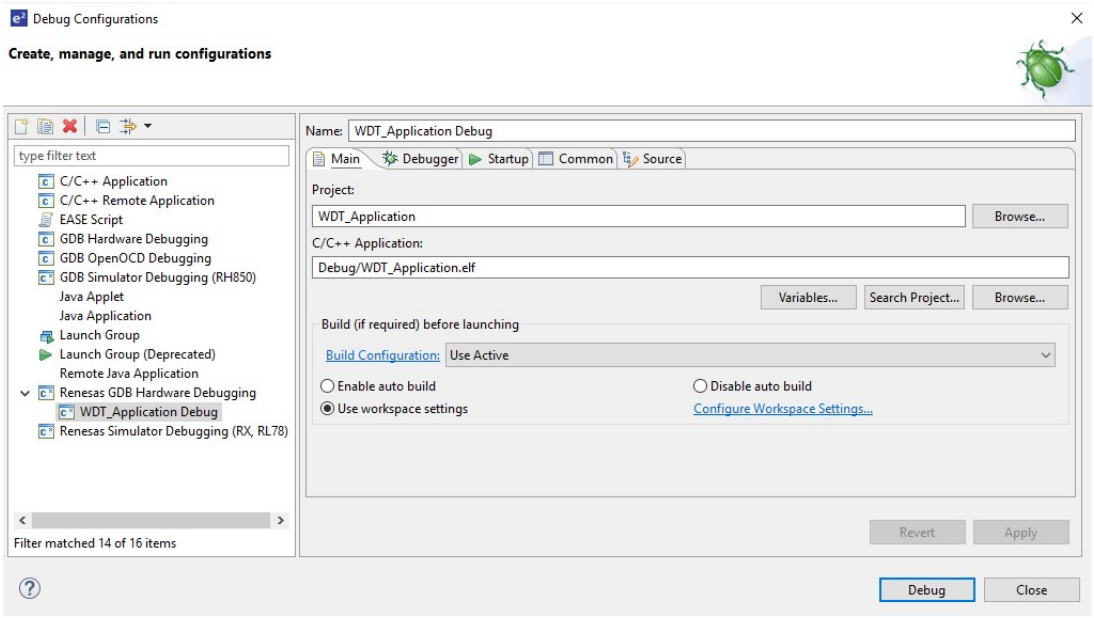
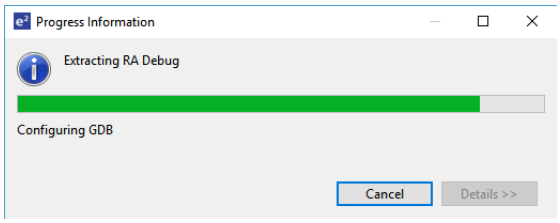



Figure 121: Debug configuration

4. Click the **Debug** button. Click Yes to the debug perspective if asked.




5. The code should run the Reset_Handler() function.
6. Resume execution via **Run > Resume**. Execution will stop in main() at the call to hal_entry().
7. Resume execution again.

The red LED should start flashing. After 30 flashes the green LED will start flashing and the red LED will stop flashing.

While the green LED is flashing the WDT will underflow and reset the device resulting in the red LED to flash again as the sequence repeats.

1. Stop the debugger in e² studio via **Run > Terminate**.
2. Click the reset button on the target board. The LEDs begin flashing.

# 3.6 Primer: Arm TrustZone Project Development

This section will introduce the user to the tools supporting Arm® TrustZone® technology configuration for the RA Family of microcontrollers. It is intended to be read by development engineers implementing RA Arm TrustZone projects for the first time. It will introduce basic concepts

followed by workflow and tooling functions designed to simplify and accelerate their first Arm TrustZone development. A background knowledge of e² studio and RA device hardware is expected.

### 3.6.1 Target Device

RA Cortex®-M33 and Cortex®-M85 devices with Arm TrustZone security extension. As shown in the following table, most RA devices that include TrustZone support also support Device Lifecycle Management (DLM).

| MCU Groups with TrustZone and DLM | MCU Groups with TrustZone and no DLM | MCU Groups with TrustZone and Alternate DLM |
|---|---|---|
| RA4E1, RA4M2, RA4M3, RA6E1, RA6M4, RA6M5, RA6T2, RA8E1, RA8E2, RA8P1, RA8T2 | RA4E2, RA4T1, RA6E2, RA6T3 | RA8M1, RA8D1, RA8T1 |

### 3.6.2 Renesas Implementation of Arm TrustZone Technology

The following section is supplied for reference only. For full details of TrustZone implementation, refer to Arm documentation (https://developer.arm.com/ip-products/security-ip/trustzone) and the MCU user manual.

Arm TrustZone technology divides the MCU and therefore the application into Secure and Non-Secure partitions. Secure applications can access both Secure and Non-Secure memory and resources. Non-Secure code can access Non-Secure memory and resources as well as Secure resources through a set of so-called veneers located in the Non-Secure Callable (NSC) region. This ensures a single access point for Secure code when called from the Non-Secure partition. The MCU starts up in the Secure partition by default. The security state of the CPU can be either Secure or Non-Secure.

The MCU code flash, data flash, and SRAM are divided into Secure (S) and Non-Secure (NS) regions. Code flash and SRAM include a further region known as Non-Secure Callable (NSC). The method to set these memory security attributes depends on whether the device supports DLM:

- For devices that support DLM, memory security attributes are set into the non-volatile memory through SCI or USB boot mode commands when the device lifecycle is Secure Software Debug (SSD) state. The memory security attributes are loaded into the Implementation Defined Attribution Unit (IDAU) peripheral and the memory controller before application execution and cannot be updated by application code.
- For devices that do not support DLM, memory security attributes are written to IDAU registers at startup by application code using secure accesses.

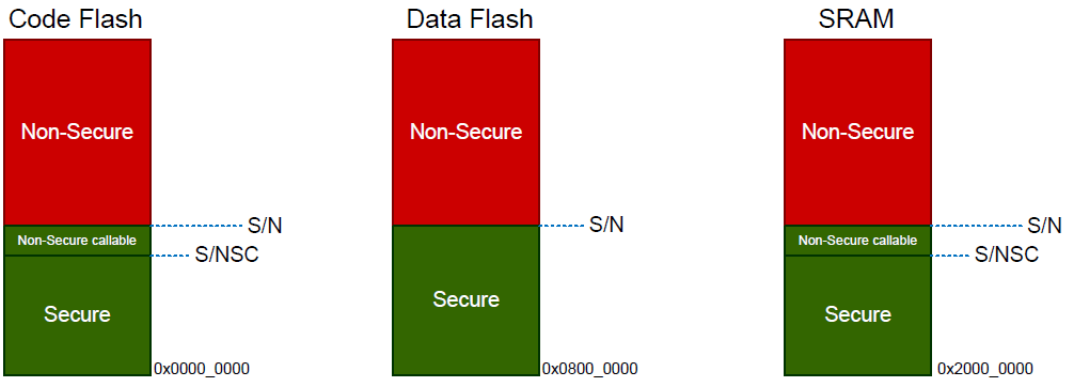



Figure 122: Secure and Non-Secure Regions

Note: All external memory accesses are considered to be Non-Secure.

Code Flash and SRAM can be divided into Secure, Non-Secure, and Non-Secure Callable. All secure memory accesses from the Non-Secure region MUST go through the Non-Secure Callable gateway and target a specific Secure Gateway (SG) assembler instruction. This forces access to Secure APIs at a fixed location and prevents calls to sub-functions and so on. Failing to target an SG instruction will generate a TrustZone exception.

TrustZone enabled compilers will manage generation of the NSC veneer automatically using CMSE extensions.

### 3.6.2.1 Calling from Non-Secure to Secure

A new instruction SG (Secure Gateway) has been added to the Armv8-M architecture. This MUST be the destination instruction for any branch within the Non-Secure Callable region. If an attempt is made to branch to any other instruction from the Non-Secure partition, a TrustZone exception will be thrown.

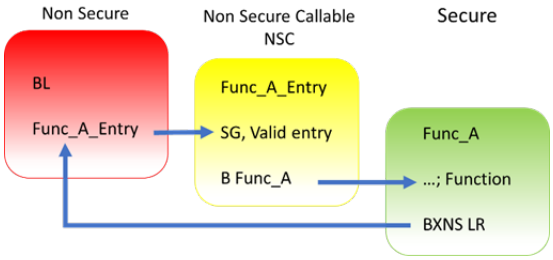



Figure 123: Calling from Non-Secure to Secure Functions

### 3.6.2.2 Calling from Secure to Non-Secure

Secure code uses B(L)XNS instructions to make direct calls to Non-Secure functions. While this is certainly possible, it can create a security vulnerability in the application. It is also challenging for the Secure application to determine the address of the non-secure function during build phase. From the RA Tools and FSP point view, calling directly from Secure to Non-Secure via FSP API is not supported.

Preference is for the Secure code to initialise as necessary from reset, then pass control to the Non-Secure partition. It will manage any data transfers and so forth via FSP call-backs as security checks. For example, secure data can be copied to Non-Secure RAM.

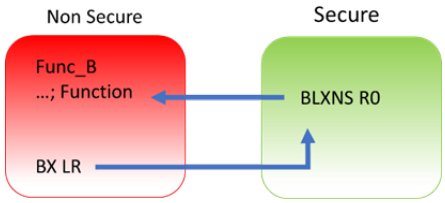



Figure 124: Calling from Secure to Non-Secure Functions

## 3.6.3 Workflow

Arm TrustZone MCU development normally consists of two projects within a workspace, Secure and Non-Secure. General project workflows are described in the following sections. The Renesas project generator also supports development with "Flat project" model with no Arm TrustZone awareness.

### 3.6.3.1 Secure Project

1. Start a new Secure project in e² studio.
2. Select and configure pins and drivers/stacks that need to be initialized and used in Secure mode. This should be kept to a minimum to reduce the security attack surface.
3. Expose top of stacks as Non-Secure Callable (NSC) ***if*** they need to be accessed from Non-Secure partition. Again, this should be kept to a minimum.
4. Generate project content and write Secure code such as key handling and opening drives as needed.
5. Modify/remove any unnecessary "Guard" functions as needed to control access via NSC.
6. Build project.
7. A Non-Secure project will be needed before debugging. If necessary, prepare a "dummy" Non-Secure project or replace R_BSP_NonSecureEnter(); with while(1); in hal_entry.c.

### 3.6.3.2 Non-Secure Project

1. Start a new Non-Secure project.
2. If you have access to the Secure project, choose this option. However, if you only have access to a device with pre-programmed Secure code (commonly referred to as provisioned device) choose "Secure Bundle".
3. Select and configure pins and drivers/stacks that need to be initialized and used in Non-Secure mode.
4. Note that you can add NSC drivers and stacks as needed.
5. Generate project content and write Non-Secure code as needed
6. Access NSC drivers and Stacks via Guard functions.
7. Build and debug project.

### 3.6.3.3 Flat Project

A flat project does not technically use Arm TrustZone as the developer has made a decision to place the entire application in Secure partition from restart.

Notes:

- Any code placed in external memory (such as OSPI or QSPI) will be Non-Secure.

- The Ethernet EDMAC is designed to be a Non-Secure bus master so associated Ethernet RAM buffers will be placed in Non-Secure RAM. The tooling will automatically manage this.

The workflow is as follows:

1. Start a new Flat project.
2. Select and configure pins and drivers/stacks as needed.
3. Generate project content and write code as needed.
4. Build and debug project.

### 3.6.4 RA Project Generator (PG)

The RA project generators have been created to help users through setting up new TrustZone enabled projects. User will be prompted for project settings such as Project Type (Secure, Non-Secure, or Flat), compiler, RTOS and debugger. Care is needed when setting up a TrustZone project to ensure that the connection between Secure and Non-Secure partitions are managed correctly.

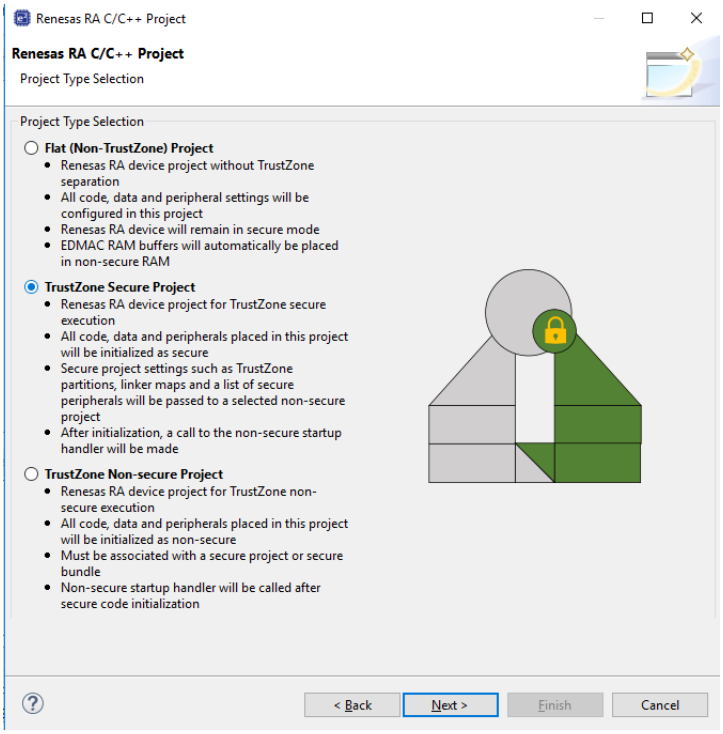



Figure 125: Secure Project (following Arm notation as green)

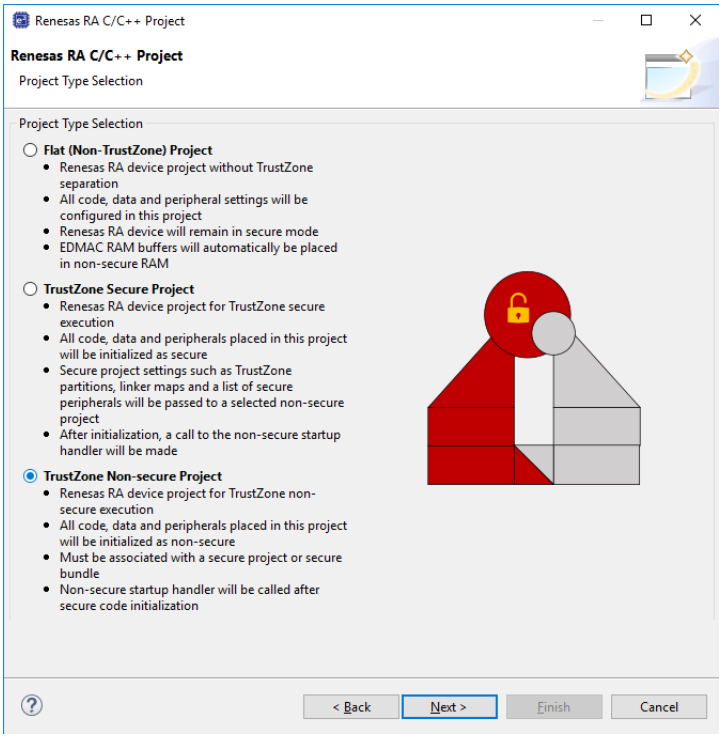



Figure 126: Non-Secure Project (following Arm notation as red)

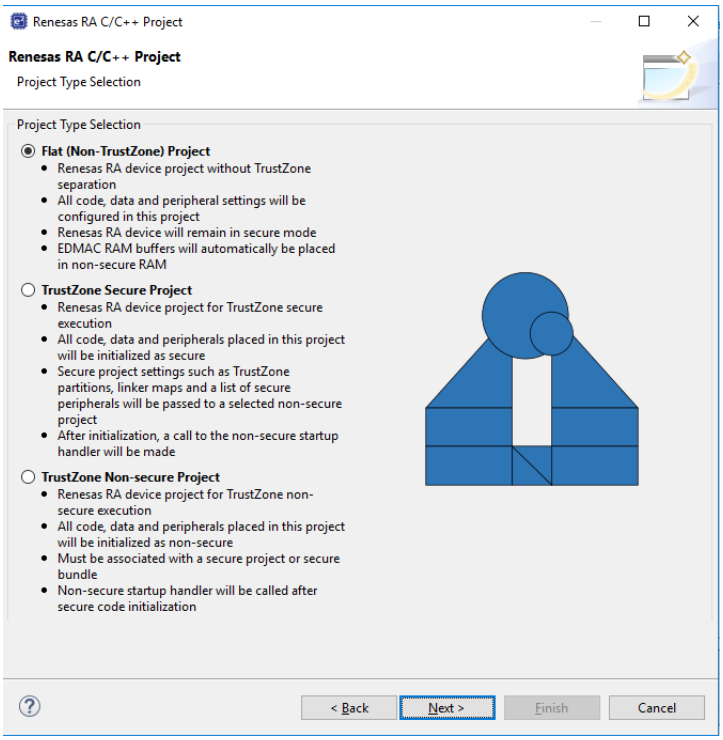



Figure 127: Flat Project

### 3.6.4.1 Secure Project Set Up

All code, data, and peripherals in this project will be configured as Secure using the device Peripheral

Security Attribution (PSA) registers. Although it is very application specific, we recommend keeping the Secure project code as small as possible to reduce the attack surface. For example, secure key handling may be the only application code in the secure project.

Necessary values to set up the TrustZone memory partition (IDAU registers) will be automatically calculated after the project is built to ensure they match the code and data size, keeping the attack surface as small as possible.

Typically, ANSI C start up code (clearing of RAM, variable initialisation, etc) , clock, and secure peripheral initialisation will occur in this project.

At the end of the Secure code, a call will be made to R_BSP_NonSecureEnter(); to pass control to the Non-Secure partition.

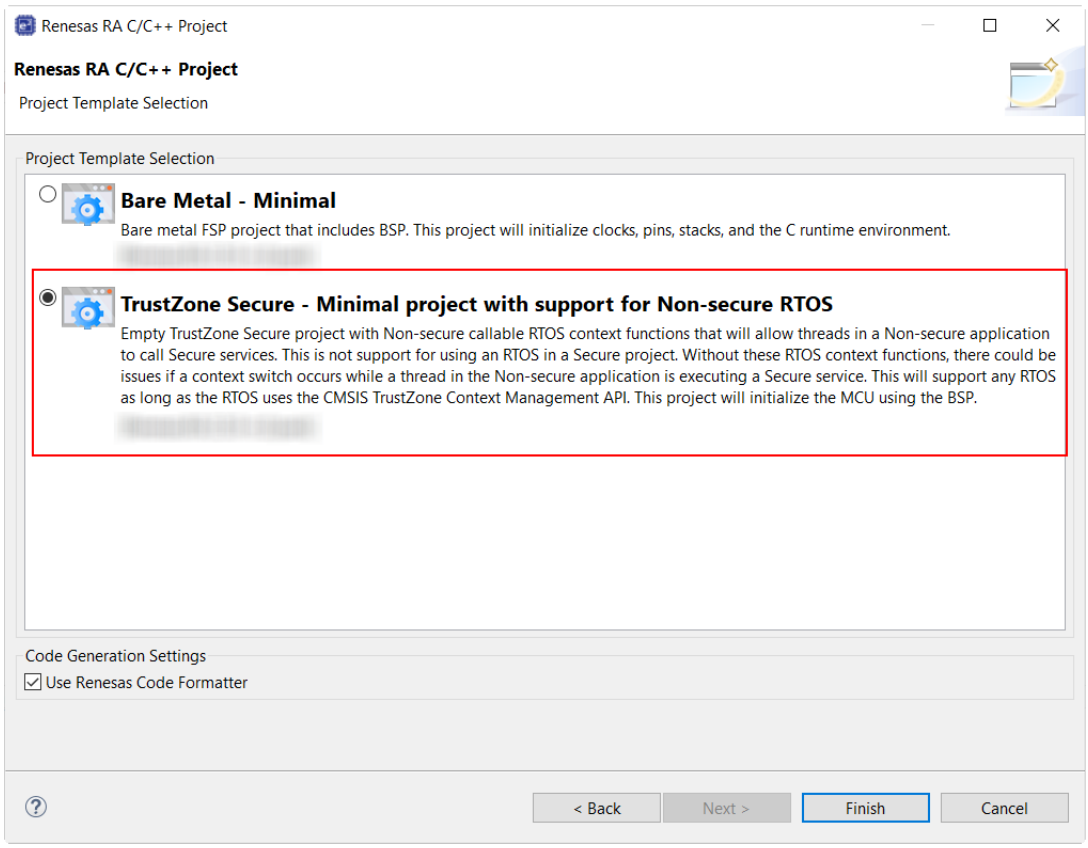
Non-Secure Callable (NSC) "Guard" functions are added to the project and expose selected modules to Non-Secure projects. User can add application-specific access checks as needed in these functions.

Output of this project type will be an elf file that must be either pre-programmed (provisioned) into a device or referenced by a Non-Secure project (via Secure bundle *.SBD) to build a final image.

This project type will NOT typically be debugged in isolation and will normally require a Non-Secure project such as a call to a R_BSP_NonSecureEnter() to be made. This can be replaced with while(1); if needed.

#### 3.6.4.2 RTOS Support in TrustZone Project

Although the RTOS kernel and user tasks will reside in the Non-Secure partition, the Secure partition needs to allocate stack space and so on. It is essential when starting a new RTOS project that the TrustZone Secure RTOS-Minimal template is selected. This will add the Arm TrustZone Context RA Port as below.



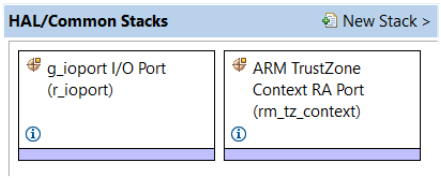




Figure 128: Secure RTOS-Minimal Template

### 3.6.4.3 Peripheral Security Attribution

Each peripheral can be configured to be Secure or Non-Secure. Peripherals are divided into two types.

Type-1 peripherals have one security attribute. Access to all registers is controlled by one security attribute. The Type-1 peripheral security attribute is set in the PSARx (x = B to E) register by the secure application.

Type-2 peripherals have the security attribute for each register or for each bit. Access to each register or bit field is controlled according to these security attributes. The Type-2 peripheral security attribute is set in the Security Attribution register in each module by the Secure application. For more information about the Security Attribution register, see sections in the Appropriate MCU's User's Manual for each peripheral.

Table 1. Secure and Non-Secure Peripherals

| Type | Peripheral |
|---|---|
| Type 1 | SCI, SPI, USBFS, CAN, IIC, SCE9, DOC, SDHI, SSIE, CTSU, CRC, CAC, TSN, ADC12, DAC12, POEG, AGT, GPT, RTC, IWDT, WDT |
| Type 2 | System control (Resets, LVD, Clock Generation Circuit, Low Power Modes, Battery Backup Function), FLASH CACHE, SRAM controller, CPU CACHE, DMAC, DTC, ICU, MPU, BUS, Security setting, ELC, I/O ports |
| Always Non-Secure | CS Area Controller, QSPI, OSPI, ETHERC, EDMAC |

FSP will initialise the arbitration registers during Secure project BSP start up. User code may also be written to set or clear further arbitration. However care must be taken not to undermine FSP.

#### 3.6.4.4 Non-Secure

All code, data, and peripherals in this project will be configured as Non-Secure. This project type must be associated with a Secure project to enable access to secure code, peripherals, linker scripts and others.

#### 3.6.4.5 Flat Project Type

All code, data, and peripherals are configured in a Secure single partition except for the EDMAC RAM buffers that will remain in the Non-Secure partition. Effectively, TrustZone is disabled.

#### 3.6.4.6 Secure Connection to Non-Secure Project

When starting a new Non-Secure Project, the user will be prompted for either a Secure Project or Secure Bundle. In each case, details of the linker settings, Non-Secure Callable functions, and Secure peripherals will be read to enable the Non-Secure project setup.

Should the Secure project or bundle be rebuilt, the Non-Secure editor will detect this and prompt user to regenerate the Non-Secure project configuration.

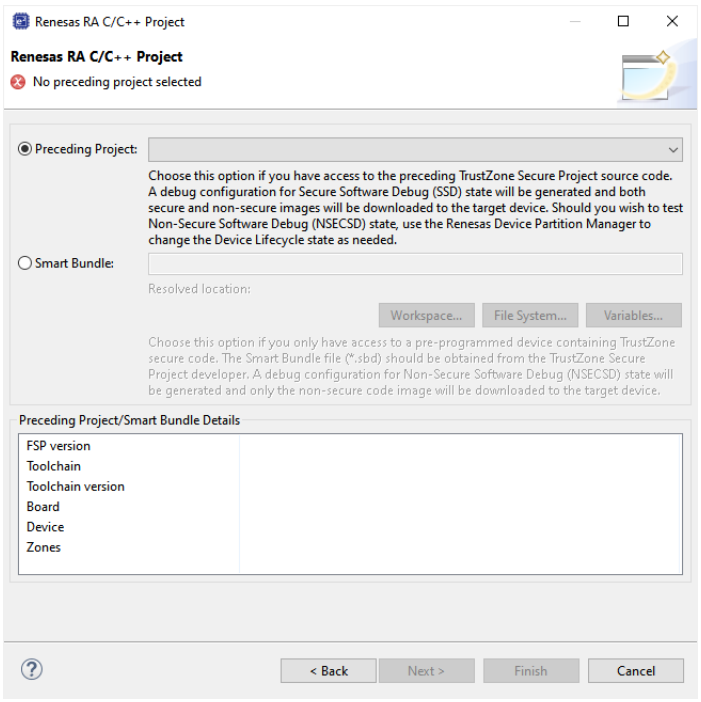



Figure 129: Secure Project or Bundle Selection

### Secure Project (Combined)

A Secure project must reside in the same Workspace as the Non-Secure project and will typically be used when a design engineer has access to both the Secure and Non-Secure project sources. This is sometimes known as "Combined model".

A Secure .elf file will be referenced and included in the debug configuration for download to the target device. The development engineer will have visibility of Secure and Non-Secure project source code and configuration.

### Secure Bundle (Split)

A Secure Bundle will ONLY include linker memory ranges, symbol references, and details of locked Secure peripheral configuration settings but no access to Secure source code (API header files will be included as necessary).

The Secure bundle file (*.SBD) must be supplied to the Non-Secure developer by the Secure project developer.

The development engineer will typically not have access to the Secure project or .elf file which MUST be pre-programmed or provisioned into the target MCU.

For devices that support DLM, the DLM state of the target device should then be switched to NSECSD (see section 6.2) before the device is provided to the non-secure developer.

This is often referred to as "Split model" where a basic security set up is developed by a Secure team and then passed to the Non-Secure team in the same facility or at a third party. The Non-Secure team has no access to the Secure source code and cannot directly access Secure peripherals, data, or APIs.

#### 3.6.4.7 Debug Configurations

After each project type has been selected, a suitable debug configuration will be generated.

**Non-Secure with Secure Project (Combined)**

Both Secure and Non-Secure .elf files will be downloaded.

A debug configuration called <project name>_SSD will be generated.

**Non-Secure with Secure Bundle (Split)**

Only a Non-Secure elf will be downloaded. This configuration must be used with a pre-provisioned device (Secure project pre-programmed into MCU Flash).

A debug configuration called <project name>_NSECSD will be generated.

**Flat Debug**

A single .elf file will be downloaded.

A debug configuration called <project name>_FLAT will be generated.

### 3.6.5 Secure Projects

As mentioned, Secure code will be called immediately after device reset and run ANSI C start up, clock, interrupt vector table, and secure peripheral initialization before starting user code. All selected peripheral configuration settings will be automatically initialised as Secure.

#### 3.6.5.1 Secure Clock

Device clock settings are the possible exception in that they will be initialised in the Secure project (to enable faster start up from reset) but can be set as Secure or Non-Secure as user application may need to change settings during execution (for low-power mode and so on). The Secure and Non-Secure FSP BSPs can both change the clock settings.

However, clock settings can be locked as Secure should the developer choose to do so.

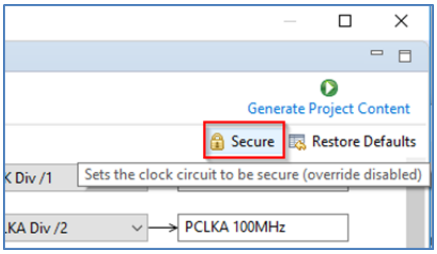



Figure 130: Secure Clock Setting

#### 3.6.5.2 Setting Drivers as NSC

Some driver and middleware stacks in the Secure project may need to be accessed by the Non-Secure partition. To enable generation of NSC veneers, set "Non-Secure Callable" from the right-click context menu for the selected modules in the Configurator.

Note: It is only possible to "expose" top of stacks as NSC.

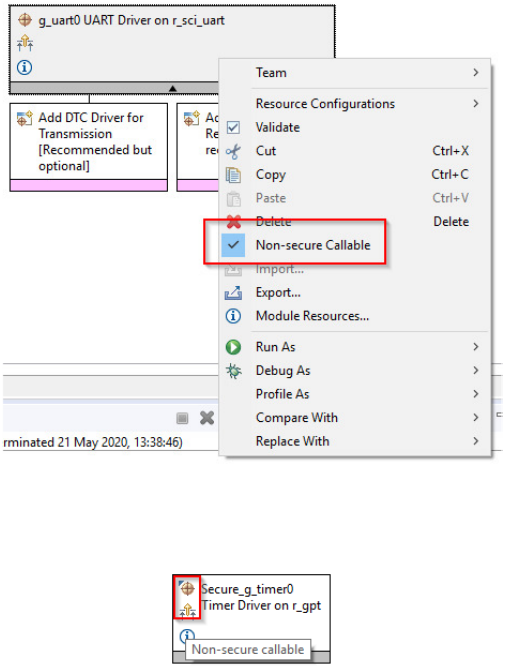



Figure 131: Generate NSC Veneers

The top of the stack will be marked with a new icon and tool tip to signify NSC access.

### 3.6.5.3 Guard Functions

Access to NSC drivers from a Non-Secure project is possible through the Guard APIs. FSP will automatically generate Guard functions for all the top of stack/driver APIs added to the project as Non-Secure Callable.

User can choose to add further levels of access control or delete guard function if they wish to only expose a limited range of APIs to a Non-Secure developer.

```
BSP_CMSE_NONSECURE_ENTRY fsp_err_t g_uart0_open_guard(
       uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) {
    /* User can add security checks here */
    FSP_PARAMETER_NOT_USED(p_api_ctrl);
    FSP_PARAMETER_NOT_USED(p_cfg);
    return R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
}
```

For example, an SCI channel may be opened and configured for a desired baud rate by the Secure developer, but only enable the Write API to the Non-Secure developer. In which case, all but g_uart0_write_guard() could be deleted. CTRL structures are not required as they will be added on the Secure side.

For example, the call from the Non-Secure partition would be as follows:

```
err = g_uart0_open_guard(0,0);
```

## 3.6.6 Non-Secure projects

Configuration of the project can continue as for other RA devices, but certain resources will be locked if they have been previously set up as Secure.

The Non-Secure project will be called from the Secure project via R_BSP_NonSecureEnter();

### 3.6.6.1 Clock Set Up

You may recall that clocks can be set as Secure or Non-Secure. If they are set as Secure, settings will only be available to view, and user will not be able to change them. The Override button will be greyed. This is useful to preserve CGC sync with secure project by not overriding unless necessary. If it is NOT set as Secure, user can choose to override the initial Secure settings

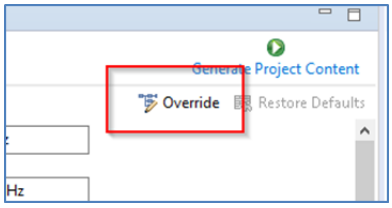



Figure 132: Clock Setting as Non-Secure

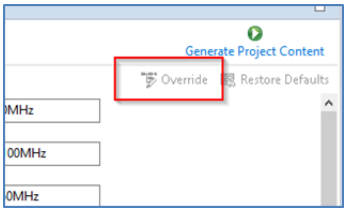
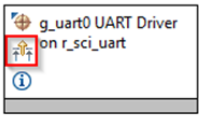



Figure 133: Clock Setting as Secure

### 3.6.6.2 Selecting NSC Drivers

Drivers declared as NSC in a Secure project can be selected and added to Non-Secure project and will be decorated as before.



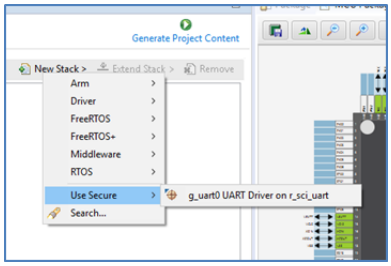



Figure 134: Selecting NSC Drivers

### 3.6.6.3 Locked Resources

When a NSC Secure driver is added to a Non-Secure project, the configuration settings are locked and are available for information only. A padlock is added for indication.

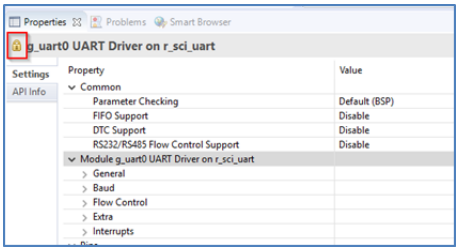



Figure 135: Locked Resources

### 3.6.6.4 Locked Channels

In a peripheral with multiple channels, for example, DMA, if a Non-Secure developer tries to select a channel that has already been defined as Secure, the following error message type will be displayed.

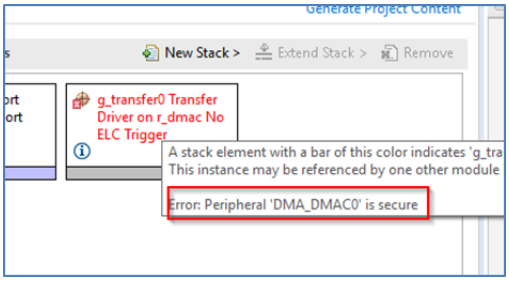



Figure 136: Error Message when Selecting a Secure Channel

## 3.6.7 IDAU registers

Renesas RA TrustZone-enabled devices include a set of registers known as Implementation Defined Attribution Unit (IDAU) that are used to set up partitions between Secure, Non-Secure Callable, and Non-Secure regions.

For devices that use DLM, the IDAU registers can only be programmed during MCU ***boot mode*** and NOT through the debug interfaces. Because of this, special debugger firmware has been developed to manage bringing the device up in SCI boot mode to set up the IDAU registers (automatically

drives MD pin) and then switch back to debug mode as needed.

**Note:** Please be aware of the extra signal connection (MD pin) needed on the debug interface connector. The Renesas Evaluation Kit (EK) for your selected device is a good reference.

| Pin No. | SWD | JTAG | Serial Programming using SCI |
|---|---|---|---|
| 1 | VCC | VCC | VCC |
| 2 | P108/SWDIO | P108/TMS | NC |
| 4 | P300/SWCLK<br>Wired OR with MD | P300/TCK<br>Wired OR with MD | P201/MD |
| 6 | P109/SWO/TXD9 | P109/TDO/TXD9 | P109/TXD9 |
| 8 | P110/RXD9 | P110/TDI/RXD9 | P110/RXD9 |
| 9 | GNDdetect | GNDdetect | GNDdetect |
| 10 | nRESET | nRESET | nRESET |
| 12 | P214/TRACECLK | P214/TRACECLK | NC |
| 14 | P211/TRACEDATA[0] | P211/TRACEDATA[0] | NC |
| 16 | P210/TRACEDATA[1] | P210/TRACEDATA[1] | NC |
| 18 | P209/TRACEDATA[2] | P209/TRACEDATA[2] | NC |
| 20 | P208/TRACEDATA[3] | P208/TRACEDATA[3] | NC |
| 3, 5, 15,17, 19 | GND | GND | GND |
| 7 | NC | NC | NC |
| 11, 13 | NC | NC | NC |

The e² studio build phase automatically extracts the IDAU partition register settings from the Secure .elf file and programs them into the device during debug connection, which can be observed in the console.

This is an important phase of TrustZone development as the Secure partitions should be set as small as possible to ensure that the security attack surface is as small as possible.

However, should the developer wish to make these partitions larger to accommodate, for example during field firmware updates, const or data arrays should be placed in the Secure project as needed.

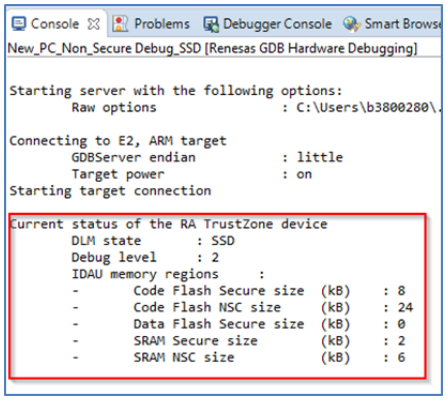



Figure 137: RA TrustZone Device Current Status

It is also possible to manually set up the partition registers through the Renesas Device Partition Manager.

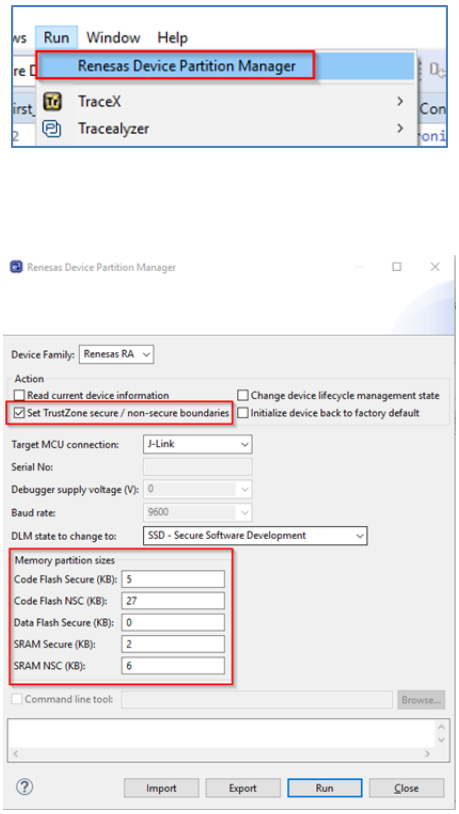
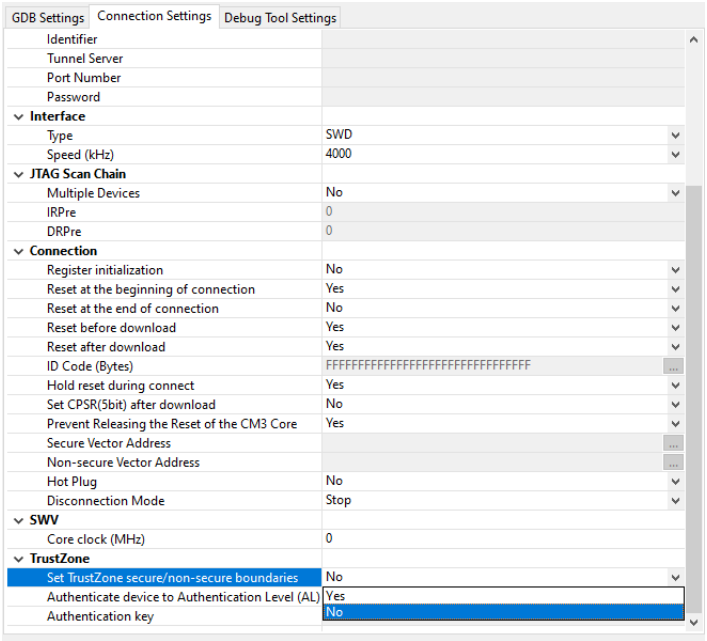



Figure 138: Renesas Device Partition Manager

In e² studio, when manually setting partitions, make sure to disable setting partitions in the debug configuration to prevent the settings from being overridden when a debug session is launched.




### 3.6.7.1 SCI Boot Mode

Example of MD mode pin connection to debugger connector (from EK schematic).

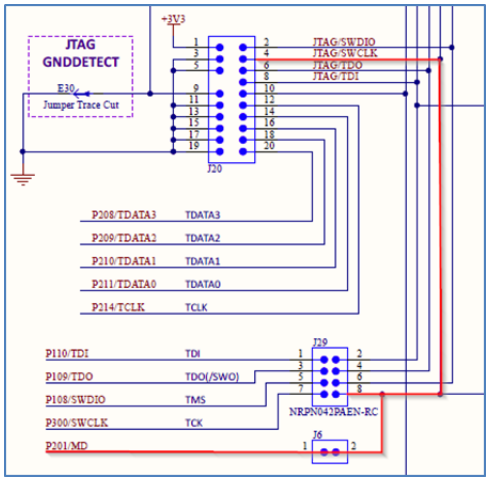



Figure 139: Example of MD Mode Pin Connection to Debugger Connector (from EK schematic

### 3.6.7.2 DLM States

Device lifecycle defines the current phase of the device and controls the capabilities of the debug interface, the serial programming interface and Renesas test mode. The following illustration shows the lifecycle definitions and capability in each lifecycle.

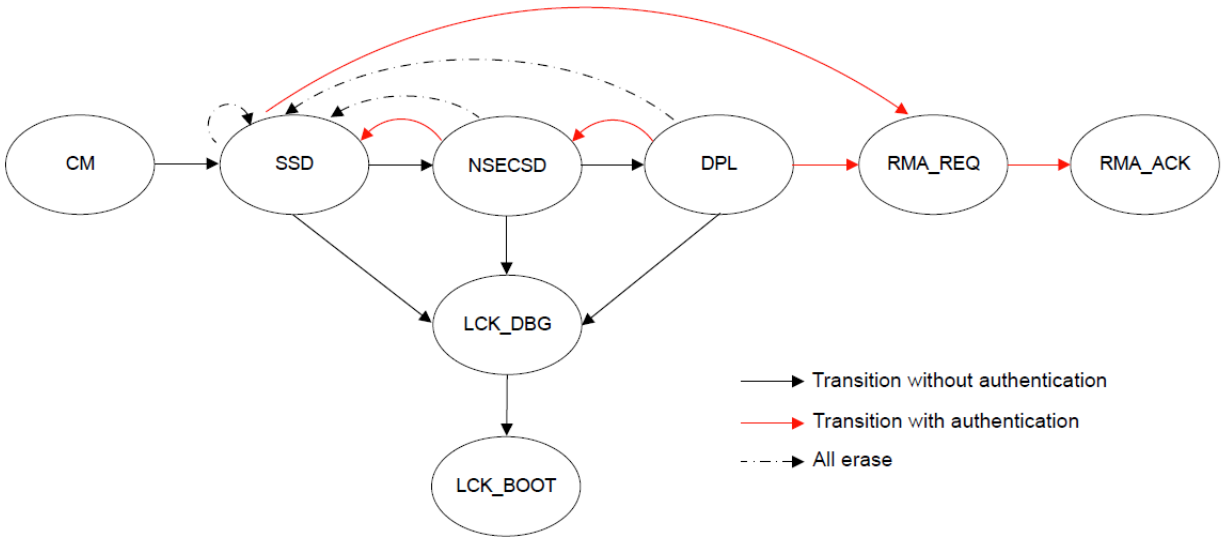



Figure 140: Lifecycle Stages

Note: All authentication key exchange and transitioning to LCK_DBG, LCK_BOOT, RMA_REQ is only managed by Renesas Flash Programmer (RFP) or other production programming tools, and NOT within e² studio.

| Lifecycle | Definition | Debug level | Serial programming | Test mode |
|---|---|---|---|---|
| CM | "Chip Manufacturing"<br>The state when the customer received the device. | DBG2 | Available, cannot access code/data flash | Not available |
| SSD | "Secure Software Development"<br>The secure part of application is being developed. | DBG2 | Available<br>can program/erase/read all code/data flash area | Not available |
| NSECSD | "Non-SECure Software Development"<br>The non-secure part of application is being developed. | DBG1 | Available<br>can program/erase/read all code/data flash area | Not available |
| DPL | "DePLoyed"<br>The device is in-field. | DBG0 | Available<br>cannot access code/data flash area | Not available |
| LCK_DBG | "LoCKed DeBuG"<br>The debug interface is permanently disabled. | DBG0 | Available<br>cannot access code/data flash area | Not available |
| LCK_BOOT | "LoCKed BOOT interface"<br>The debug interface and the serial programming interface are permanently disabled. | DBG0 | Not available | Not available |
| RMA_REQ | "Return Material Authorization REQuest"<br>Request for RMA. The customer must send the device to Renesas in this state. | DBG0 | Available<br>cannot access code/data flash area | Not available |
| RMA_ACK | "Return Material Authorization ACKnowledged"<br>Failure analysis in Renesas | DBG2 | Available<br>cannot access code/data flash area | Available |

Figure 141: Lifecycle Stages and Debug Levels

There are three debug access levels. The debug access level changes according to the lifecycle state.

- DBG2: The debugger connection is allowed, and no restriction to access memories and peripherals
- DBG1: The debugger connection is allowed, and restricted to access only Non-Secure memory regions and peripherals
- DBG0: The debugger connection is not allowed

State transitions can be performed using the Renesas Flash Programmer (RFP, see section below) or using the Renesas Device Partition Manager (limited number of states possible). It is possible to secure transitions between states using authentication keys. For more information on DLM states and transitions (device specific), please refer to device user manual.

### 3.6.7.3 Devices with Alternate DLM States

Some devices have an alternate implementation of Device Lifecycle Management (See Target Device ). While the concept is the same, the DLM states are a little different. The following illustration shows the lifecycle definitions and capability in each lifecycle.

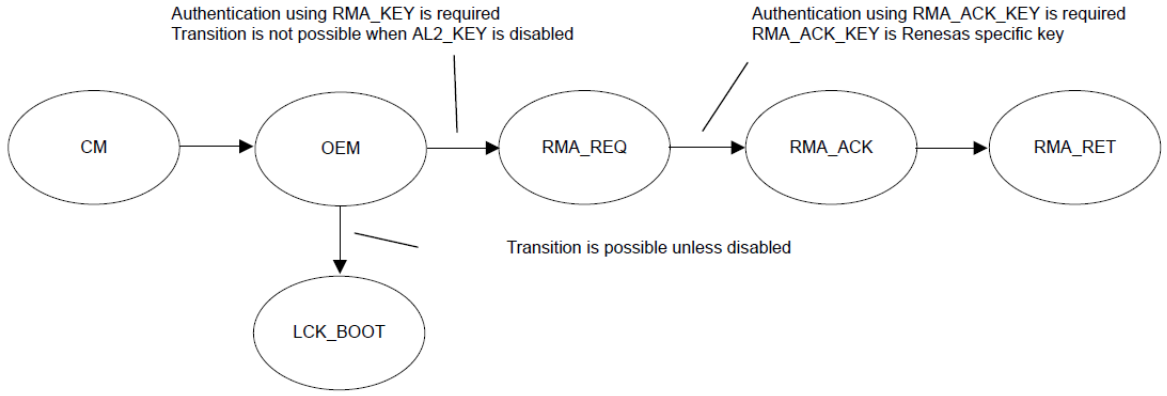



Figure 142: Lifecycle Stages

Note: All authentication key exchange and transitioning to LCK_BOOT, and RMA_REQ is only managed by Renesas Flash Programmer (RFP) or other production programming tools, and NOT within e² studio.

| Lifecycle | Definition | Protection level | Debug function | Serial programming | Renesas test mode |
|---|---|---|---|---|---|
| CM | "Chip Manufacturing"<br>The device is out of Renesas factory.<br>The customer receives the device in this state. | PL2 | Available in the secure and non-secure debug | Available<br>Cannot access code/data flash area | Not available |
| OEM | "Original Equipment Manufacturer"<br>The device is owned by the customer. | PL2 or PL1 or PL0 | Depend on the authentication level | | Not available |
| LCK_BOOT | "LoCKed BOOT interface"<br>The debug interface and the serial programming interface are permanently disabled. | PL0 | Not available | Not available | Not available |
| RMA_REQ | "Return Material Authorization REQuest"<br>Request for RMA. The customer must send the device to Renesas in this state. | PL0 | Not available | Available<br>Cannot access code/data flash area | Not available |
| RMA_ACK | "Return Material Authorization ACKnowledged"<br>Failure analysis in Renesas | PL2 | Available in the secure and non-secure debug | Available<br>Cannot access code/data flash area | Available |
| RMA_RET | "Return Material Authorization RETurn"<br>The device is back to the customer. The device does not boot. | PL0 | Not available | Not available | Not available |

Figure 143: Lifecycle Stages and Debug Levels

There are three authentication levels. The available authentication levels change according to the lifecycle state and determine the memory and resources that are accessible by the debugger.

- AL2: The debugger connection is allowed, and no restriction to access memories and peripherals
- AL1: The debugger connection is allowed, and restricted to access only Non-Secure memory regions and peripherals
- AL0: The debugger connection is not allowed

State transitions can be performed using the Renesas Flash Programmer (RFP, see section below) or using the Renesas Device Partition Manager (limited number of states possible). It is possible to secure transitions between states using authentication keys. For more information on DLM states and transitions (device specific), please refer to device user manual.

In addition to having different DLM States, these devices also handle memory partitioning differently. Code Flash and Data Flash partitions are divided into secure and non-secure regions using SCI or USB boot mode commands. All other memory regions are set in IDAU and SAU registers in the secure application during the BSP startup procedure. All memory regions and peripherals that are configured as non-secure must be accessed using an aliased non-secure address. This non-secure address is calculated by setting bit 28 in the address.

#### 3.6.7.4 Devices without DLM

For devices that do not have DLM, the IDAU registers are programmed by secure application code at startup. Devices without DLM do not support IDAU register programming using boot mode commands.

## 3.6.8 Debug

By default, devices supporting DLM will be in SSD mode and so allow access to Secure and Non-Secure partitions. In this mode both Secure and Non-Secure .elf files will be downloaded.

The current debugger status is displayed in the lower left corner and includes the DLM state (SSD or NSECSD) and current partition (Secure, Non-Secure, or Non-Secure Callable) when the debugger is stopped, for example.

Figure 144: Current Debugger Status

#### 3.6.8.1 Non-Secure Debug

Once the device is transitioned to NSECSD mode, only Non-Secure Flash, RAM and Peripherals can be accessed. In this mode, a Secure .elf must be pre-programmed (provisioned) into the device, and only a Non-Secure .elf file will be downloaded.

When in NSECSD mode access to Secure elements will be blocked and data displayed as ????????.

In NSECSD mode, it is not possible to set breakpoints on Secure code or data.

It is not possible to step into Secure code; the debugger will perform a step-over of any Secure function calls. Should the user press the Suspend button during execution, the debugger will stop at the next Non-Secure code access.

Assuming Secure memory region finishes at 32K (0x8000) in NSECSD debug mode (colour coding

added for indication only), memory will be displayed as shown in the following figure.

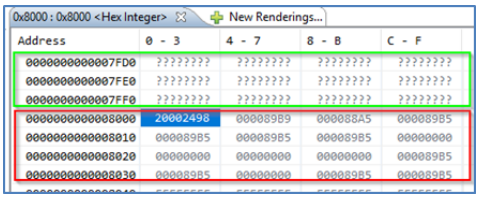



Figure 145: Memory Display in NSECSD Debug Mode

Disassembly will be displayed as shown in the following figure.

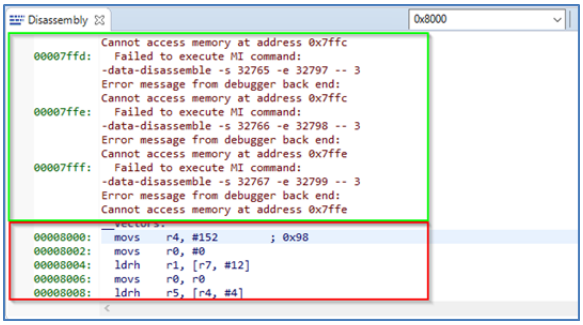



Figure 146: Disassembly Display in NSECSD Debug Mode

## 3.6.9 Debugger support

Renesas E2, E2 Lite, and SEGGER J-Link are supported in e² studio for TrustZone projects.

**Debugger Support for TrustZone Projects**

| Feature | E2 Lite | E2 | J-Link | J-Link OB | ULINK | IAR I-jet |
|---|---|---|---|---|---|---|
| JTAG | Yes | Yes | Yes | No | Yes | Yes |
| SWD | Yes | Yes | Yes | Yes | Yes | Yes |
| ETB trace | Yes | Yes | Yes | Yes | Yes | Yes |
| ETM trace | No | Yes | Yes | No | Yes | Yes |
| TrustZone partition programming | Yes | Yes | Yes | Yes | No | Yes |
| Non secure debug | Yes | Yes | Yes | Yes | No | Yes |
| e² studio | Yes | Yes | Yes | Yes | No | No |
| IAR EW Arm | Yes | Yes | Yes | Yes | No | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| Keil MDK | Under consideration | Under consideration | Yes | Yes | Yes | No |

## 3.6.10 Third-Party IDEs

Third-party IDEs such as IAR Systems EWARM and Keil MDK (uVision) are supported by the RA Smart Configurator (RASC).

In general, RASC offers the same configurator functionality as e² studio documented above. Project generators are available to initialise workspaces in the target IDEs as well as setting up debug configurations and so forth. However, there are some limitations that need to be noted especially with regards to IDAU TrustZone partition register programming. See the specific RASC documentation for usage details.

## 3.6.11 Renesas Flash Programmer (RFP)

Updated versions of Renesas Flash Programmer (RFP) are available to support setting of partitions, DLM state and Authentication keys.

RFP can be downloaded free of charge on the Renesas web site.

A new mode has been added to Program Flash Options as shown in the following graphics.

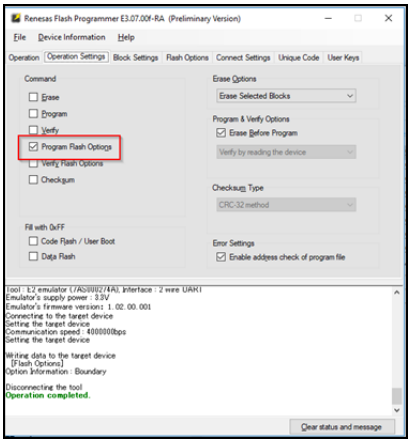



Figure 147: RFP Program Flash Options

Options to set partition boundaries are shown in the following figure.

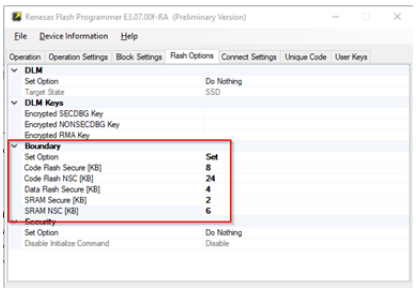



Figure 148: RFP Partition Boundaries

Options to set DLM state, Authentication keys, and Security settings are shown in the following figure.

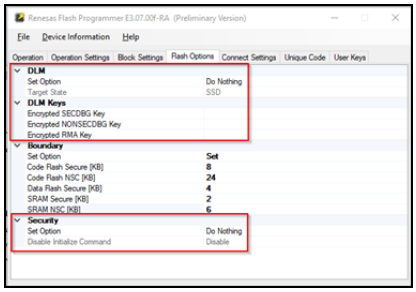



Figure 149: RFP DLM State, Authentication Keys, and Security Settings

Great care is needed here as some DLM states can **permanently** turn off debug and boot mode on the devices. Equally programming a security access authentication key can lead to permanently locked devices if the key is lost.

## 3.6.12 Glossary

**IDAU**

Implementation Defined Attribute Unit. Used to program TrustZone partitions.

**NSECSD**

Non-Secure Software Development mode

**SSD**

Secure Software Development mode

**NSC**

Non-Secure Callable. Special Secure memory region used for Veneer to allow access to Secure APIs from Non-Secure code.

**Provisioned**

Device with Secure code pre-programmed and DLM state set to **NSECSD**

**Flat project**

All code, data and peripherals are configured as secure with the exception of the EDMAC RAM buffer which are placed in Non-Secure RAM due to the configuration of the internal bus masters.

**Veneer**

Code that resides in Non-Secure Callable region

**Combined model**

Development engineer has access to both Secure and Non-Secure project and source code

**Split model**

Development Engineer has access to only the Non-Secure partition. No visibility of Secure source code. Secure code will be provisioned into device.

#### 3.6.12.1 Configurator Icon Glossary

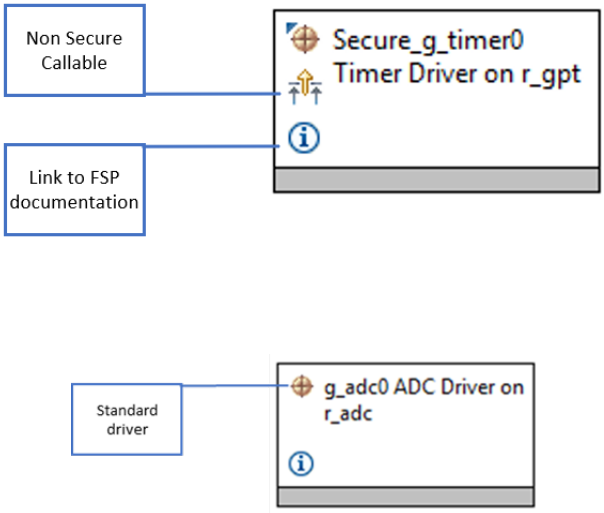



Figure 150: Configurator Icons

# 3.7 RASC User Guide for MDK and IAR

## 3.7.1 What is RASC?

The Renesas RA Smart Configurator (RASC) is a desktop application designed to configure device hardware such as clock set up and pin assignment as well as initialization of FSP software components for a Renesas RA microcontroller project when using a 3rd-party IDE and toolchain.

The RA Smart Configurator can currently be used with

1. Keil MDK and the Arm compiler toolchain.
2. IAR EWARM with IAR toolchain for Arm

Projects can be configured and the project content generated in the same way as in e² studio. Please refer to Configuring a Project section for more details.

## 3.7.2 Using RA Smart Configurator with Keil MDK

#### 3.7.2.1 Prerequisites

- Keil MDK and Arm compiler are installed and licensed. Please refer to the RASC Release notes for the version to be installed.
- Import the RA device pack. Download the RA device pack archive file (ex: MDK_Device_Packs_2.x.x.zip) from the FSP GitHub release page. Extract the archive file to locate the RA device pack. To import the RA device pack, launch the PackInstaller.exe from <keil_mdk_install_dir>\UV4. Select the menu item **File > Import...** and browse to the

extracted .pack file.
- Verify that the latest updates for RA devices are included in Keil MDK. To verify, select the menu "Packs" in Pack Installer and verify that the menu item **Check for Updates on Launch** is selected. If not, select **Check for Updates on Launch** and relaunch Pack Installer.
- For flashing and debugging, the latest Segger J-Link DLL is installed into Keil MDK.
- Install RASC and FSP using the Platform Installer from the GitHub release page.

#### 3.7.2.2 Create new RA project

The following steps are required to create an RA project using Keil MDK, RASC and FSP:

1. Start the RA Smart Configurator.

2. Enter a project folder and project name.

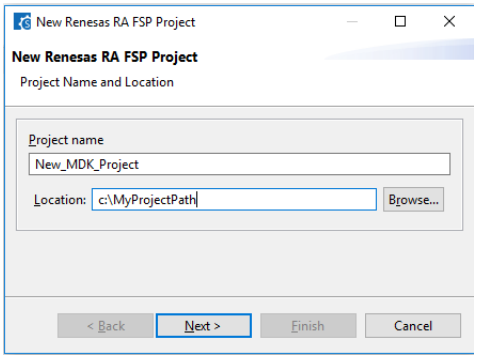



Figure 151: RASC project settings

3. Select the target device and IDE.

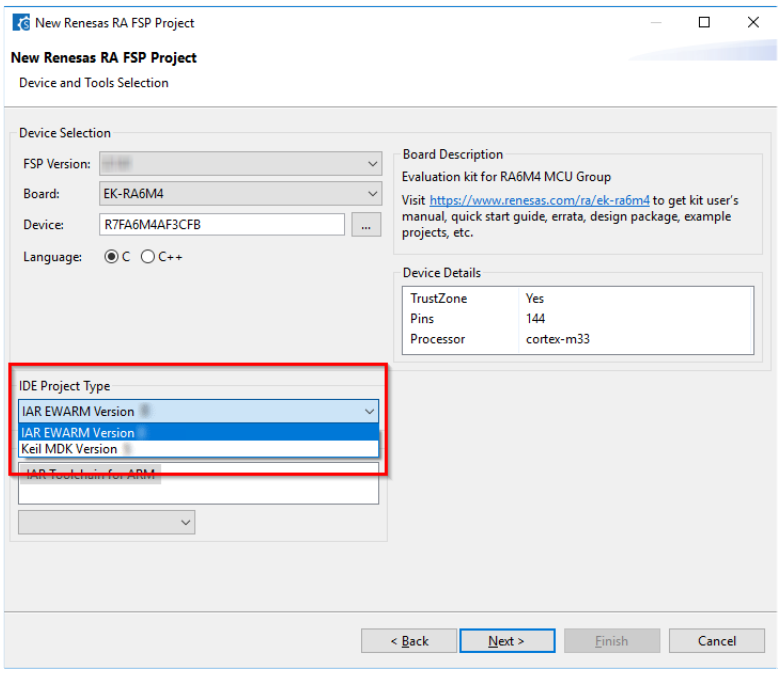



Figure 152: Target device and IDE selection

4. The rest of the project generator and FSP configuration is the same as e² studio. Please refer to the previous sections for details.
5. On completion of FSP configuration, press "Generate Project Content"

6. A new Keil MDK project file will be generated in the project path. Double click this file to open MDK and continue development as usual.

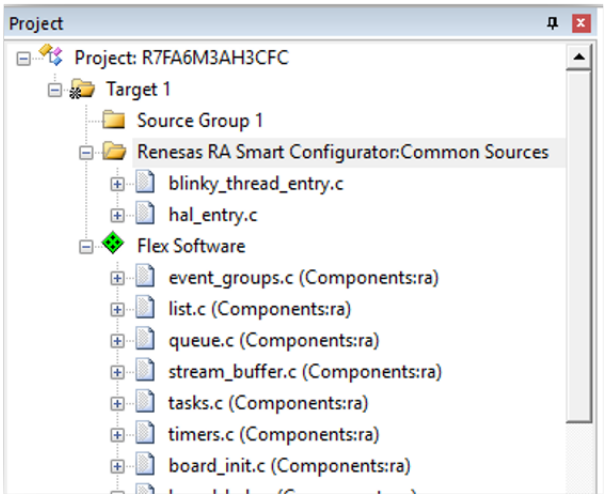



Figure 153: uVision project workspace with imported project data

### 3.7.2.3 Modify existing RA project

Once an initial project has been generated and configured, it is also possible to make changes using RASC as follows.

*Note*

*This setup only needs to be done once per project.*

Set up the following links to RASC:

1. In Keil MDK uVision, select **Tools > Customize Tools Menu...**.
2. Select the **new** icon and fill in the fields as follows for each tool:
    a. RA Smart Configurator:
        - Menu item name: Enter: RA Smart Configurator
        - Command: Select "..." and navigate to rasc.exe
        - Initial Folder: Enter: $P
        - Arguments: Enter: configuration.xml

    b. Device Partition Manager:
        - Menu item name: Enter: Device Partition Manager
        - Command: Select "..." and navigate to rasc.exe
        - Initial Folder: Enter: $P
        - Arguments: Enter: -application com.renesas.cdt.ddsc.dpm.ui.dpmapplication configuration.xml "$L%L"

To reconfigure an existing project select **Tools > RA Smart Configurator**

To reconfigure the TrustZone partitions select **Tools > Device Partition Manager**

### 3.7.2.4 Build and Debug RA project

The project can be built by selecting the menu item **Project > Build Target** or tool bar item **Rebuild** or the keyboard shortcut F7.

Assembler, Compiler, Linker and Debugger settings can be changed in **Options for Target** dialog, which can be launched using the menu item **Project > Options for Target**, the tool bar item **Options for Target** or the keyboard shortcut Alt+F7.

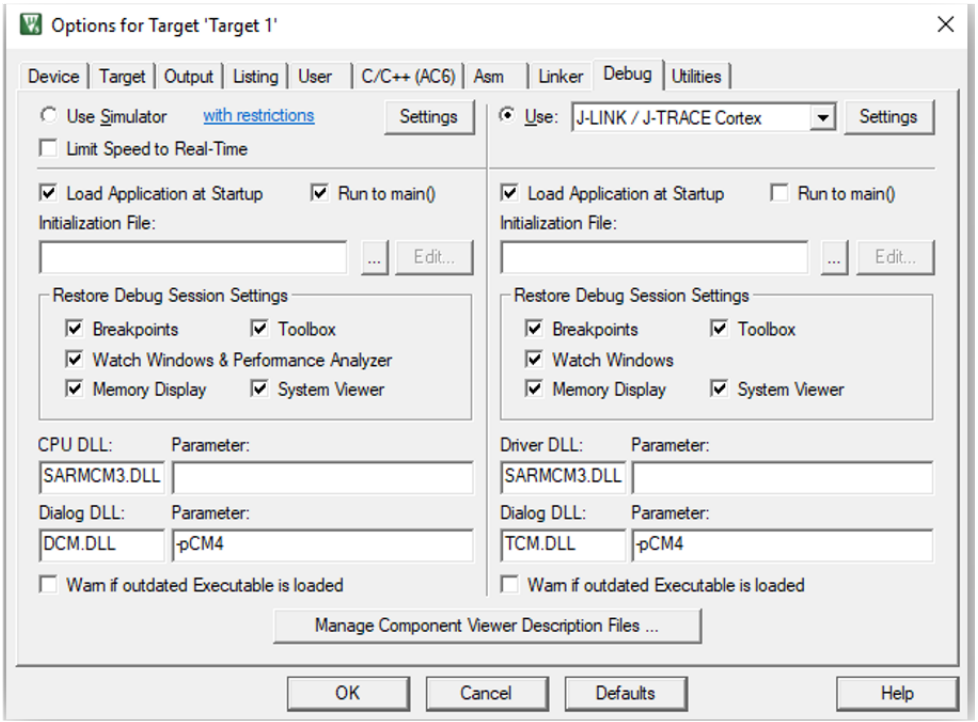



Figure 154: Options for Target

RASC will set up the uVision project to debug the selected device using J-Link or J-Link OB debugger by default.

A Debug session can be started or stopped by selecting the menu item **Debug > Start/Stop Debug Session** or keyboard shortcut CTRL+F5. When debugging for the first time, J-Link firmware update may be needed if requested by the tool.

Refer to the documentation from Keil to get more information on the debug features in uVision. Note that not all features supported by uVision debugger are implemented in the J-Link interface. Consult SEGGER J-Link documentation for more information.

#### 3.7.2.5 Notes and Restrictions

1. **When debugging a TrustZone based project, the Secure project image MUST be downloaded before the Non Secure project.**

2. For TrustZone enabled devices, the user will need to manually set up the memory partitions using the "Renesas Device Partition Manager" from inside RASC before downloading.

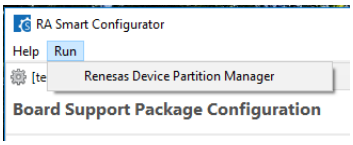



Figure 155: Renesas Device partition Manager

3. RA FSP contains a full set of drivers and middleware and may not be compatible with other CMSIS packs from Keil, Arm or third parties.
4. Flash programming is currently only supported through the debugger connection.

### 3.7.3 Using RA Smart Configurator with IAR EWARM

IAR Systems Embedded Workbench for Arm (EWARM) includes support for Renesas RA devices. These can be set up as bare metal designs within EWARM. However, most RA developers will want to integrate RA FSP drivers and middleware into their designs. RASC will facilitate this.

RASC generates a "Project Connection" file that can be loaded directly into EWARM to update project files.

#### 3.7.3.1 Prerequisites

- IAR EWARM installed and licensed. Pleae refer to the Release notes for the version to be installed.
- RASC and FSP Installed

#### 3.7.3.2 Create new RA project

The following steps are required to create an RA project using IAR EWARM, RASC and FSP:

1. Start the RA Smart Configurator.

2. Enter a project folder and project name.

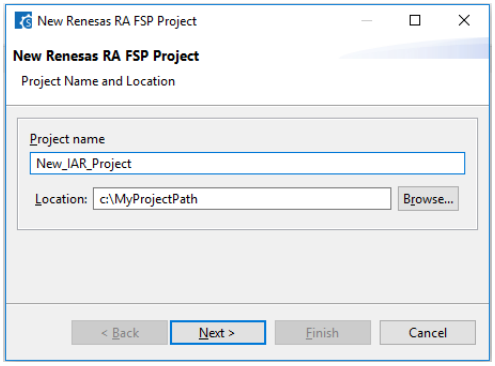



Figure 156: RASC project settings

3. Select the target device and IDE.

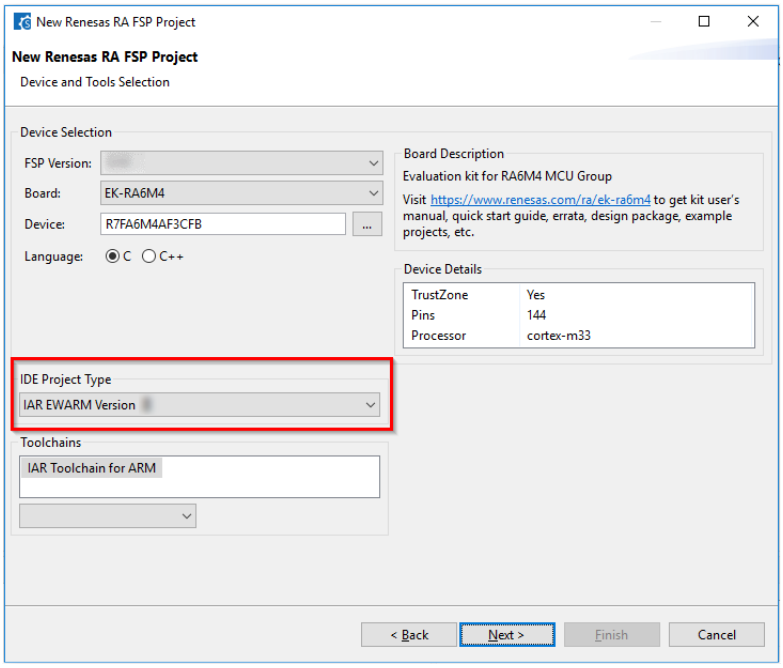



Figure 157: Target device and IDE selection

4. The rest of the project generator and FSP configuration operates the same as e² studio. Refer to the previous sections for details.
5. On completion of FSP configuration, press **Generate Project Content**.
6. A new IAR EWARM project file will be generated in the project path. Double click this file to open IAR EWARM and continue development as usual.
7. To Use RASC with EWARM, RASC needs to configured as a tool in EWARM by selecting the menu item **Tools > Configure Tools...**. Select **New** to create a new tool in the dialog shown and add the following information:
    - Menu Text: **RA Smart Configurator**
        a. Command: Select **Browse...** and navigate to rasc.exe in the installed RASC
        b. Argument: configuration.xml

c. Initial Directory: $PROJ_DIR$
d. Tool Available: Always

- Menu Text: **Device Partition Manager**
  a. Command: Select **Browse...** and navigate to rasc.exe in the installed RASC
  b. Argument: -application com.renesas.cdt.ddsc.dpm.ui.dpmapplication configuration.xml "$TARGET_PATH$"
  c. Initial Directory: $PROJ_DIR$
  d. Tool Available: Always

8. RASC can now be re-launched from EWARM using the menu item **Tools > RA Smart Configurator**.
9. A Project connection needs to be set up in EWARM to build the project. Select **Project > Add Project Connection** in EWARM and select **IAR Project Connection**. Navigate to the project folder and select buildinfo.ipcf and click **Open**. The project can now build in EWARM.

#### 3.7.3.3 Notes and Restrictions

When starting a TrustZone enabled debug session Partition sizes are checked automatically.

- If partition sizes are set correctly, the debug session will launch as normal.
- If partition sizes need to be changed, IAR EWARM will prompt to run the Renesas Device Partition Manager. Select **Yes**. The Device Partition Manager will start with the required partition sizes prefilled.
- Select **Set TrustZone secure / non-secure boundaries** as the only action.
- Enter debugger details, if required.
- Select **Run** to program the partitions.
- Return to the IDE and relaunch the debug session

# Chapter 4 FSP Architecture

## 4.1 FSP Architecture Overview

This guide describes the Renesas Flexible Software Package (FSP) architecture and how to use the FSP Application Programming Interface (API).

### 4.1.1 C99 Use

FSP uses the ISO/IEC 9899:1999 (C99) C programming language standard. Specific features introduced in C99 that are used include standard integer types (stdint.h), booleans (stdbool.h), designated initializers, and the ability to intermingle declarations and code.

### 4.1.2 Doxygen

Doxygen is the default documentation tool used by FSP. You can find Doxygen comments throughout the FSP source.

### 4.1.3 Weak Symbols

Weak symbols are used occasionally in FSP. They are used to ensure that a project builds even when the user has not defined an optional function.

### 4.1.4 Memory Allocation

Dynamic memory allocation through use of the malloc() and free() functions are not used in FSP modules; all memory required by FSP modules is allocated in the application and passed to the module in a pointer. Exceptions are considered only for ports of 3rd party code that require dynamic memory.

### 4.1.5 FSP Terms

| Term | Description | Reference |
|---|---|---|
| BSP | Short for Board Support Package. In FSP, the BSP provides just enough foundation to allow other FSP modules to work together without issue. | MCU Board Support Package |

| | | |
|---|---|---|
| Module | Modules can be peripheral drivers, purely software, or anything in between. Each module consists of a folder with source code, documentation, and anything else that the customer needs to use the code effectively. Modules are independent units, but they may depend on other modules. Applications can be built by combining multiple modules to provide the user with the features they need. | FSP Modules |
| Driver | A driver is a specific kind of module that directly modifies registers on the MCU. | - |
| Interface | An interface contains API definitions that can be shared by modules with similar features. Interfaces are definitions only and do not add to code size. | FSP Interfaces |
| Stacks | The FSP architecture is designed such that modules work together to form a stack. A stack consists of a top level module and all its dependencies. | FSP Stacks |
| Module Instance | Single and independent instantiation of a module. An application may require two GPT timers. Each of these timers is a module instance of the r_gpt module. | - |
| Application | Code that is owned and maintained by the user. Application code may be based on sample application code provided by Renesas, but it is the responsibility of the user to maintain as necessary. | - |

| | | |
|---|---|---|
| Callback Function | This term refers to a function that is called when an event occurs. As an example, suppose the user would like to be notified every second based on the RTC. As part of the RTC configuration, a callback function can be supplied that will be jumped to during each RTC interrupt. When a single callback services multiple events, the arguments contain the triggering event. Callback functions for interrupts should be kept short and handled carefully because when they are called the MCU is still inside of an interrupt, delaying any pending interrupts. | - |

# 4.2 FSP Modules

Modules are the core building block of FSP. Modules can do many different things, but all modules share the basic concept of providing functionality upwards and requiring functionality from below.

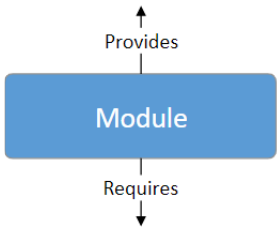



Figure 158: Modules

The amount of functionality provided by a module is determined based on functional use cases. Common functionality required by multiple modules is often placed into a self-contained submodule so it can be reused. Code size, speed and complexity are also considered when defining a module.

The simplest FSP application consists of one module with the Board Support Package (BSP) and the user application on top.

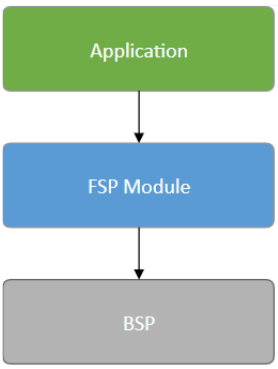



Figure 159: Module with application

The Board Support Package (BSP) is the foundation for FSP modules, providing functionality to determine the MCU used as well as configuring clocks, interrupts and pins. For the sake of clarity, the BSP will be omitted from further diagrams.

### 4.2.1 Module Sources

Some modules distributed alongside FSP originate from outside sources. A full list of sources for FSP modules, including versions and hyperlinks, can be found in the Third Party Software section of the release notes for each release.

### 4.2.2 Module Distribution

All modules distributed with FSP are packaged as CMSIS components in CMSIS packs. Each module consists of source files and a tooling support file used to integrate the module with e² studio or RASC. The tooling support file defines the configurations used to generate code in the ra_gen and ra_cfg folders.

### 4.2.3 Module Versioning

Module versions can be seen on the Components tab of the FSP Configuration editor. The FSP Configuration editor automatically selects compatible components.

All third party modules have a semantic version are versioned with their original semantic version plus added metadata fsp.<fsp_semantic_version>. The metadata is added to reflect the tooling support file added for the FSP configuration tool.

Third party modules versioned with +renesas.<counter> in the metadata have been forked and updated for FSP. If +renesas.<counter> is not in the metadata, the third party code is unchanged from its original source.

If changes are made to third party module source code to support FSP, the changes are pushed to a public Renesas GitHub fork of the original source. Links to Renesas forks are provided in the Third Party Software section of the release notes for each release.

Modules that originate from outside sources that do not have a semantic version are versioned with the FSP version.

All modules that are part of FSP or integrated with FSP are tested as a package. Mixing versions is not encouraged and may lead to support issues.

# 4.3 FSP Stacks

When modules are layered atop one another, an FSP stack is formed. The stacking process is performed by matching what one module provides with what another module requires. For example, the SPI module (SPI (r_spi)) requires a module that provides the transfer interface (Transfer Interface ) to send or receive data without a CPU interrupt. The transfer interface requirement can be fulfilled by the DTC driver module (Transfer (r_dtc)).

Through this methodology the same code can be shared by several modules simultaneously. The example below illustrates how the same DTC module can be used with SPI (SPI (r_spi)), UART (UART (r_sci_uart)) and SDHI (SD/MMC (r_sdhi)).

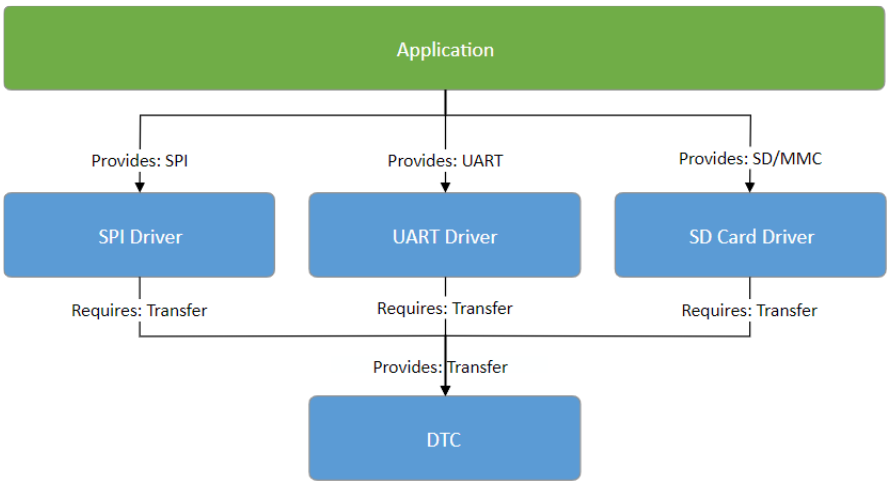



Figure 160: Stacks -- Shared DTC Module

The ability to stack modules ensures the flexibility of the architecture as a whole. If multiple modules include the same functionality issues arise when application features must work across different user designs. To ensure that modules are reusable, any dependent modules must be capable of being swapped out for other modules that provide the same features. The FSP architecture provides this flexibility to swap modules in and out through the use of FSP interfaces.

# 4.4 FSP Interfaces

At the architecture level, interfaces are the way that modules provide common features. This commonality allows modules that adhere to the same interface to be used interchangeably. Interfaces can be thought of as a contract between two modules - the modules agree to work together using the information that was established in the contract.

On RA hardware there is occasionally an overlap of features between different peripherals. For example, I2C communications can be achieved through use of the IIC peripheral or the SCI peripheral. However, there is a difference in the level of features provided by both peripherals; in I2C mode the SCI peripheral will only support a subset of the capabilities of the fully-featured IIC.

Interfaces aim to provide support for the common features that most users would expect. This means that some of the advanced features of a peripheral (such as IIC) might not be available in the interface. In most cases these features are still available through interface extensions.

In FSP design, interfaces are defined in header files. All interface header files are located in the folder

ra/fsp/inc/api and end with *_api.h. Interface extensions are defined in header files in the folder ra/fsp/inc/instances. The following sections detail what makes up an interface.

### 4.4.1 FSP Interface Enumerations

Whenever possible, interfaces use typed enumerations for function parameters and structure members.

```
typedef enum e_i2c_master_addr_mode
{
    I2C_MASTER_ADDR_MODE_7BIT = 1,    ///< Use 7-bit addressing mode
    I2C_MASTER_ADDR_MODE_10BIT = 2,    ///< Use 10-bit addressing mode
} i2c_master_addr_mode_t;
```

Enumerations remove uncertainty when deciding what values are available for a parameter. FSP enumeration options follow a strict naming convention where the name of the type is prefixed on the available options. Combining the naming convention with the autocomplete feature available in e² studio (Ctrl + Space) provides the benefits of rapid coding while maintaining high readability.

### 4.4.2 FSP Interface Callback Functions

Callback functions allow modules to asynchronously alert the user application when an event has occurred, such as when a byte has been received over a UART channel or an IRQ pin is toggled. FSP driver modules define and handle the interrupt service routines for RA MCU peripherals to ensure any required hardware procedures are implemented. The interrupt service routines in FSP modules then call the user-defined callbacks to allow the application to respond.

Callback functions must be defined in the user application. They always return void and take a structure for their one parameter. The structure is defined in the interface for the module and is named <interface>_callback_args_t. The contents of the structure may vary depending on the interface, but two members are common: event and p_context.

The event member is an enumeration defined in the interface used by the application to determine why the callback was called. Using the UART example, the callback could be triggered for many different reasons, including when a byte is received, all bytes have been transmitted, or a framing error has occurred. The event member allows the application to determine which of these three events has occurred and handle it appropriately.

The p_context member is used for providing user-specified data to the callback function. In many cases a callback function is shared between multiple channels or module instances; when the callback occurs, the code handling the callback needs context information so that it can determine which module instance the callback is for. For example, if the callback wanted to make an FSP API call in the callback, then at a minimum the callback will need a reference to the relevant control structure. To make this easy, the user can provide a pointer to the control structure as the p_context. When the callback occurs, the control structure is passed in the p_context element of the callback structure.

Callback functions are called from within an interrupt service routine. For this reason callback functions should be kept as short as possible so they do not affect the real time performance of the user's system. An example skeleton function for the flash interface callback is shown below.

```
void flash_callback (flash_callback_args_t * p_args)
{
 /* See what event caused this callback. */
 switch (p_args->event)
    {
 case FLASH_EVENT_ERASE_COMPLETE:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_WRITE_COMPLETE:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_BLANK:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_NOT_BLANK:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_ERR_DF_ACCESS:
      {
 /* Handle error. */
 break;
       }
 case FLASH_EVENT_ERR_CF_ACCESS:
      {
 /* Handle error. */
 break;
```

```
      }
 case FLASH_EVENT_ERR_CMD_LOCKED:
     {
 /* Handle error. */
 break;
      }
 case FLASH_EVENT_ERR_FAILURE:
     {
 /* Handle error. */
 break;
      }
 case FLASH_EVENT_ERR_ONE_BIT:
     {
 /* Handle error. */
 break;
      }
 case FLASH_EVENT_ERR_TWO_BIT:
     {
 /* Handle error. */
 break;
      }
 case FLASH_EVENT_ERR_ECC:
     {
 /* Handle error. */
 break;
      }
   }
}
```

When a module is not directly used in the user application (that is, it is not the top layer of the stack), its callback function will be handled by the module above. For example, if a module requires a UART interface module the upper layer module will control and use the UART's callback function. In this case the user would not need to create a callback function for the UART module in their application code.

### 4.4.3 FSP Interface Data Structures

At a minimum, all FSP interfaces include three data structures: a configuration structure, an API structure, and an instance structure.

#### 4.4.3.1 FSP Interface Configuration Structure

The configuration structure is used for the initial configuration of a module during the <MODULE>_Open() call. The structure consists of members such as channel number, bitrate, and operating mode.

The configuration structure is used purely as an input into the module. It may be stored and referenced by the module, so the configuration structure and anything it references must persist as long as the module is open.

The configuration structure is allocated for each module instance in files generated by the RA Configuration editor.

When FSP stacks are used, it is also important to understand that configuration structures only have members that apply to the current interface. If multiple layers in the same stack define the same configuration parameters then it becomes difficult to know where to modify the option. For example, the baud rate for a UART is only defined in the UART module instance. Any modules that use the UART interface rely on the baud rate being provided in the UART module instance and do not offer it in their own configuration structures.

#### 4.4.3.2 FSP Interface API Structure

All interfaces include an API structure which contains function pointers for all the supported interface functions. An example structure for the DAC is shown below.

```
typedef struct st_dac_api
{
    /** Initial configuration.
    *
    * @param[in] p_ctrl Pointer to control block. Must be declared by user. Elements
set here.
    * @param[in] p_cfg Pointer to configuration structure. All elements of this
structure must be set by user.
    */
    fsp_err_t (* open)(dac_ctrl_t * const p_ctrl, dac_cfg_t const * const p_cfg);
    /** Close the D/A Converter.
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* close)(dac_ctrl_t * const p_ctrl);
    /** Write sample value to the D/A Converter.
```

```
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    * @param[in] value Sample value to be written to the D/A Converter.
    */
    fsp_err_t (* write)(dac_ctrl_t * const p_ctrl, uint16_t value);
    /** Start the D/A Converter if it has not been started yet.
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* start)(dac_ctrl_t * const p_ctrl);
    /** Stop the D/A Converter if the converter is running.
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* stop)(dac_ctrl_t * const p_ctrl);
} dac_api_t;
```

The API structure is what allows for modules to easily be swapped in and out for other modules that are instances of the same interface. Let's look at an example application using the DAC interface above.

RA MCUs have an internal DAC peripheral. If the DAC API structure in the DAC interface is not used the application can make calls directly into the module. In the example below the application is making calls to the R_DAC_Write() function which is provided in the r_dac module.

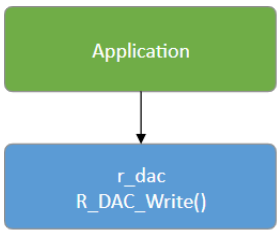



Figure 161: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU and decides to add an external DAC module named dac_external using I2C for communications. The application must now distinguish between the two modules, adding complexity and further

dependencies to the application.

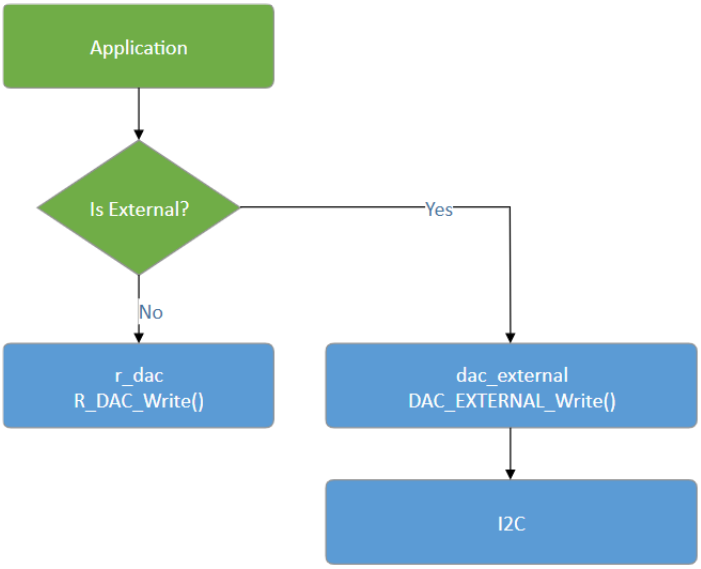



Figure 162: DAC Write with two write modules

The use of interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed if the user's dac_external module implements the FSP DAC interface, so the application no longer depends upon hard-coded module function names. Instead the application now depends on the DAC interface API which can be implemented by any number of modules.

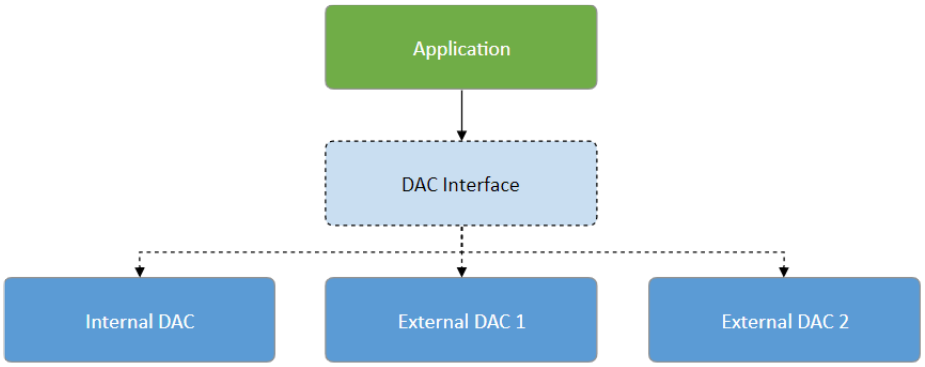



Figure 163: DAC Interface

#### 4.4.3.3 FSP Interface Instance Structure

Every FSP interface also has an instance structure. The instance structure encapsulates everything required to use the module:

- A pointer to the instance API structure (FSP Instance API)
- A pointer to the configuration structure
- A pointer to the control structure

The instance structure is not required at the application layer. It is used to connect modules to their dependencies (other than the BSP).

Instance structures have a standardized name of <interface>_instance_t. An example from the Transfer Interface is shown below.

```
typedef struct st_transfer_instance
{
  transfer_ctrl_t     * p_ctrl; ///< Pointer to the control structure for this
instance
    transfer_cfg_t const * p_cfg;      ///< Pointer to the configuration structure
for this instance
    transfer_api_t const * p_api;      ///< Pointer to the API structure for this
instance
} transfer_instance_t;
```

Note that when an instance structure variable is declared, the API is the only thing that is instance specific, not *module instance* specific. This is because all module instances of the same module share the same underlying module source code. If SPI is being used on SCI channels 0 and 2 then both module instances use the same API while the configuration and control structures are typically different.

# 4.5 FSP Instances

While interfaces dictate the features that are provided, instances actually implement those features. Each instance is tied to a specific interface. Instances use the enumerations, data structures, and API prototypes from the interface. This allows an application that uses an interface to swap out the instance when needed.

On RA MCUs some peripherals are used to implement multiple interfaces. In the example below the IIC and SPI peripherals map to only one interface each while the SCI peripheral implements three interfaces.

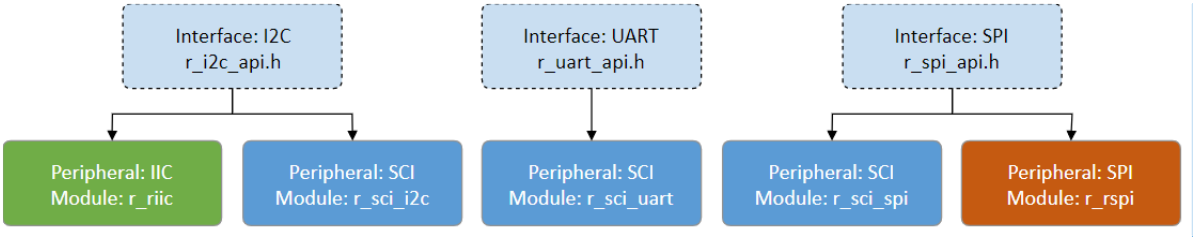



Figure 164: Instances

In FSP design, instances consist of the interface extension and API defined in the instance header file located in the folder ra/fsp/inc/instances and the module source ra/fsp/src/<module>.

## 4.5.1 FSP Instance Control Structure

The control structure is used as a unique identifier for the module instance and contains memory

required by the module. Elements in the control structure are owned by the module and *must not be modified* by the application. The user allocates storage for a control structure, often as a global variable, then sends a pointer to it into the <MODULE>_Open() call for a module. At this point, the module initializes the structure as needed. The user must then send in a pointer to the control structure for all subsequent module calls.

### 4.5.2 FSP Interface Extensions

In some cases, instances require more information than is provided in the interface. This situation can occur in the following cases:

- An instance offers extra features that are not common to most instances of the interface. An example of this is the start source selection of the GPT (Timer, General PWM (r_gpt)). The GPT can be configured to start based on hardware events such as a falling edge on a trigger pin. This feature is not common to all timers, so it is included in the GPT instance.
- An interface must be very generic out of necessity. As an interface becomes more generic, the number of possible instances increases. An example of an interface that must be generic is a block media interface that abstracts functions required by a file system. Possible instances include SD card, SPI Flash, SDRAM, USB, and many more.

The p_extend member provides this extension function.

Use of interface extensions is not always necessary. Some instances do not offer an extension since all functionality is provided in the interface. In these cases the p_extend member can be set to NULL. The documentation for each instance indicates whether an interface extension is available and whether it is mandatory or optional.

#### 4.5.2.1 FSP Extended Configuration Structure

When extended configuration is required it can be supplied through the p_extend parameter of the interface configuration structure.

The extended configuration structure is part of the instance, but it is also still considered to be part of the configuration structure. All usage notes about the configuration structure described in FSP Interface Configuration Structure apply to the extended configuration structure as well.

The extended configuration structure and all typed structures and enumerations required to define it make up the interface extension.

### 4.5.3 FSP Instance API

Each instance includes a constant global variable tying the interface API functions to the functions provided by the module. The name of this structure is standardized as g_<interface>_on_<instance>. Examples include g_spi_on_spi, g_transfer_on_dtc, and g_adc_on_adc. This structure is available to be used through an extern in the instance header file (r_spi.h, r_dtc.h, and r_adc.h respectively).

## 4.6 FSP API Standards

### 4.6.1 FSP Function Names

FSP functions start with the uppercase module name (<MODULE>). All modules have <MODULE>_Open() and <MODULE>_Close() functions. The <MODULE>_Open() function must be called before any of the other functions.

Other functions that will commonly be found are <MODULE>_Read(), <MODULE>_Write(), <MODULE>_InfoGet(), and <MODULE>_StatusGet(). The <MODULE>_StatusGet() function provides a status that could change asynchronously, while <MODULE>_InfoGet() provides information that cannot change after open or can only be updated by API calls. Example function names include:

- R_SPI_Read(), R_SPI_Write(), R_SPI_WriteRead()
- R_SDHI_StatusGet()
- R_RTC_CalendarAlarmSet(), R_RTC_CalendarAlarmGet()
- R_FLASH_HP_AccessWindowSet(), R_FLASH_HP_AccessWindowClear()

### 4.6.2 Use of const in API parameters

The const qualifier is used with API parameters whenever possible. An example case is shown below.

```
fsp_err_t R_FLASH_HP_Open(flash_ctrl_t * const p_api_ctrl, flash_cfg_t const * const
p_cfg);
```

In this example, flash_cfg_t is a structure of configuration parameters for the r_flash_hp module. The parameter p_cfg is a pointer to this structure. The first const qualifier on p_cfg ensures the flash_cfg_t structure cannot be modified by R_FLASH_HP_Open(). This allows the structure to be allocated as a const variable and stored in ROM instead of RAM.

The const qualifier after the pointer star for both p_ctrl and p_cfg ensures the FSP function does not modify the input pointer addresses. While not fool-proof by any means this does provide some extra checking inside the FSP code to ensure that arguments that should not be altered are treated as such.

### 4.6.3 FSP Version Information

The BSP provides a function R_FSP_VersionGet() which fills in a structure of type fsp_pack_version_t. This can be used to determine the FSP version at runtime.

There are also FSP_VERSION_* macros in fsp_version.h that can be used to determine the FSP version at build time.

## 4.7 FSP Build Time Configurations

All modules have a build-time configuration header file. Most configuration options are supplied at run time, though options that are rarely used or apply to all instances of a module may be moved to build time. The advantage of using a build-time configuration option is to potentially reduce code size reduction by removing an unused feature.

All modules have a build time option to enable or disable parameter checking for the module. FSP modules check function arguments for validity when possible, though this feature is disabled by default to reduce code size. Enabling it can help catch parameter errors during development and debugging. By default, each module's parameter checking configuration inherits the BSP parameter checking setting (set on the BSP tab of the RA Configuration editor). Leaving each module's parameter checking configuration set to Default (BSP) allows parameter checking to be enabled or disabled globally in all FSP code through the parameter checking setting on the BSP tab.

If an error condition can reasonably be avoided it is only checked in a section of code that can be

disabled by disabling parameter checking. Most FSP APIs can only return FSP_SUCCESS if parameter checking is disabled. An example of an error that cannot be reasonably avoided is the "bus busy" error that occurs when another master is using an I2C bus. This type of error can be returned even if parameter checking is disabled.

# 4.8 FSP File Structure

The high-level file structure of an FSP project is shown below.

```
ra_gen
ra
+---fsp
    +---inc
    |   +---api
    |   \---instances
    \---src
       +---bsp
       \---r_module
ra_cfg
+---fsp_cfg
    +---bsp
    +---driver
```

Directly underneath the base ra folder the folders are split into the source and include folders. Include folders are kept separate from the source for easy browsing and easy setup of include paths.

The ra_gen folder contains code generated by the RA Configuration editor. This includes global variables for the control structure and configuration structure for each module.

The ra_cfg folder is where configuration header files are stored for each module. See FSP Build Time Configurations for information on what is provided in these header files.

# 4.9 FSP TrustZone Support

TrustZone support for FSP is primarily handled in the RA Configuration Tool.

## 4.9.1 FSP TrustZone Projects

During development of a TrustZone project, users create an RA TrustZone Secure Project first, followed by an RA TrustZone Non-secure Project that is linked to the RA TrustZone Secure Project. Allocation of secure memory is handled automatically within the tooling. The non-secure project starts at the required alignment boundary beyond the memory taken by the secure project.

## 4.9.2 Non-Secure Callable Guard Functions

The tooling generates guard functions for any module marked as Non-secure Callable. These guard functions are owned by the application once generated, so they can be modified as necessary by the secure application developer.

The default non-secure callable guard functions limit the configuration and control structure to the structures generated in the secure project. They also check any input pointers to ensure the caller does not overwrite secure memory.

### 4.9.3 Callbacks in Non-Secure from Non-Secure Callable Modules

If the non-secure project needs a callback function from a non-secure callable module, the callback can be registered after the module is opened using the callback_set() guard function.

### 4.9.4 Migrating TrustZone Project to newer FSP Version

The TrustZone projects can be migrated to newer FSP version as mentioned in the following resource.

- Migrating Projects to New FSP Version (Application Note)

Additional steps are required if newer FSP version introduces new guard function. In such case, simply migrating the project would result in the build failure for non secure project. Following extra steps are required:

- If xxx_guard.c file in src folder of secure project was not modified earlier
    1. Delete the xxx_guard.c file in secure project before generating the Project Files.
    2. Generate Project contents and build the secure project.
    3. Follow the steps in Application Note to migrate the Non-Secure project.
- If xxx_guard.c file in src folder of secure project was modified earlier
    1. Take the backup of existing xxx_guard.c file and delete it from src folder of secure project before generating the Project Files.
    2. Generate Project contents. It creates a new guard.c file. Compare the contents of the older file and newly generated guard.c file.
    3. Copy the modified code from the older file (i.e Security checks added by user) and add it to newly generated file.
    4. Follow the steps in Application Note to migrate the Non-Secure project.

### 4.9.5 Additional TrustZone Information

The following resources provide technical background, application notes and example projects that demonstrate key TrustZone concepts and implementation procedures.

- The Benefits of Using Arm® TrustZone® in Your Design (Brochure)
- RA Arm® TrustZone® Tooling Primer (Application Note)
- Renesas RA Family Security Design with Arm® TrustZone® - IP Protection (Application Note)
- Renesas RA Family Securing Data at Rest Using the Arm® TrustZone® (Application Note)

## 4.10 FSP Architecture in Practice

### 4.10.1 FSP Connecting Layers

FSP modules are meant to be both reusable and stackable. It is important to remember that modules

are not dependent upon other modules, but upon other interfaces. The user is then free to fulfill the interface using the instance that best fits their needs.

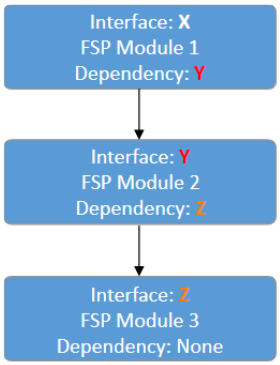



Figure 165: Connecting layers

In the image above interface Y is a dependency of interface X and has its own dependency on interface Z. Interface X only has a dependency on interface Y. Interface X has no knowledge of interface Z. This is a requirement for ensuring that layers can easily be swapped out.

## 4.10.2 Using FSP Modules in an Application

The typical use of an FSP module involves generating required module data then using the API in the application.

### 4.10.2.1 Create a Module Instance in the RA Configuration Editor

The RA Configuration editor (available both in the Renesas e² studio IDE as well as through the standalone RA Smart Configurator) provides a graphical user interface for setting the parameters of the interface and instance configuration structures. It also automatically includes those structures (once they are configured in the GUI) in application-specific header files that can be included in application code.

The RA Configuration editor allocates storage for the control structures, all required configuration structures, and the instance structure in generated files in the ra_gen folder. Use the **Properties** window to set the values for the members of the configuration structures as needed. Refer to the Configuration section of the module usage notes for documentation about the configuration options.

If the interface has a callback function option then the application must declare and define the function. The return value is always of type void and the parameter to the function is a typed structure of name <interface>_callback_args_t. Once the function has been defined, assign its name to the p_callback member of the configuration structure. Callback function names can be assigned through the **Properties** window for the selected module.

### 4.10.2.2 Use the Instance API in the Application

Call the module's <MODULE>_Open() function. Pass pointers to the generated control structure and configuration structure. The names of these structures are based on the 'Name' field provided in the configuration editor. The control structure is <Name>_ctrl and the configuration structure is <Name>_cfg. An example <MODULE>_Open() call for an r_rtc module instance named g_clock is:

```
R_RTC_Open(&g_clock_ctrl, &g_clock_cfg);
```

*Note*

*Each layer in the FSP Stack is responsible for calling the API functions of its dependencies. This means that users are only responsible for calling the API functions at the layer at which they are interfacing. Using the example above of a SPI module with a DTC dependency, the application uses only SPI APIs. The application starts by calling R_SPI_Open(). Internally, the SPI module opens the DTC. It locates R_DTC_Open() by accessing the dependent transfer interface function pointers from the pointers DTC instances (spi_cfg_t::p_transfer_tx and spi_cfg_t::p_transfer_rx) to open the DTC.*

Refer to the module usage notes for example code to help get started with any particular module.

# Chapter 5 API Reference

This section includes the FSP API Reference for the Module and Interface level functions.

| | |
|---|---|
| ▼BSP | Common code shared by FSP drivers |
| BSP I/O access | This module provides basic read/write access to port pins |
| BSP OSPI_B support | Code that initializes the OSPI_B device memory |
| BSP SDRAM support | Code that initializes the SDRAMC and SDR SDRAM device memory |
| Common Error Codes | |
| ►MCU Board Support Package | The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor |
| ▼Modules | Modules are the smallest unit of software available in FSP. Each module implements one interface |
| ►Analog | Analog Modules |
| ►AI | Artificial Intelligence Modules |
| ►Audio | Audio Modules |
| ►Bootloader | Bootloader Modules |
| ►CapTouch | CapTouch Modules |
| ►Connectivity | Connectivity Modules |
| ►DSP | DSP Modules |
| ►Graphics | Graphics Modules |
| ►Input | Input Modules |
| ►Monitoring | Monitoring Modules |
| ►Motor | Motor Modules |
| ►Networking | Networking Modules |
| ►Power | Power Modules |
| ►RTOS | RTOS Modules |
| ►Security | Security Modules |
| ►Sensor | Sensor Modules |
| ►Storage | Storage Modules |

| | |
|---|---|
| ►System | System Modules |
| ►Timers | Timers Modules |
| ►Transfer | Transfer Modules |
| ►TrustZone | Arm TrustZone Modules |
| ▼Interfaces | FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer |
| ►Analog | Analog Interfaces |
| ►AI | AI Interfaces |
| ►Audio | Audio Interfaces |
| ►CapTouch | CapTouch Interfaces |
| ►Connectivity | Connectivity Interfaces |
| ►DSP | DSP Interfaces |
| ►Graphics | Graphics Interfaces |
| ►Input | Input Interfaces |
| ►Monitoring | Monitoring Interfaces |
| ►Motor | Motor Interfaces |
| ►Networking | Networking Interfaces |
| ►Power | Power Interfaces |
| ►Security | Security Interfaces |
| ►Sensor | Sensor Interfaces |
| ►Storage | Storage Interfaces |
| ►System | System Interfaces |
| ►Timers | Timers Interfaces |
| ►Transfer | Transfer Interfaces |

# 5.1 BSP

### Detailed Description

Common code shared by FSP drivers.

### Modules

| | |
|---|---|
| | BSP I/O access<br>This module provides basic read/write access to port pins. |
| | BSP OSPI_B support<br>Code that initializes the OSPI_B device memory. |
| | BSP SDRAM support<br>Code that initializes the SDRAMC and SDR SDRAM device memory. |
| | Common Error Codes |
| | MCU Board Support Package<br>The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor. |

## Data Structures

| | |
|---|---|
| union | fsp_pack_version_t |

## Macros

| | |
|---|---|
| #define | FSP_VERSION_MAJOR |
| #define | FSP_VERSION_MINOR |
| #define | FSP_VERSION_PATCH |
| #define | FSP_VERSION_BUILD |
| #define | FSP_VERSION_STRING |
| #define | FSP_VERSION_BUILD_STRING |

## Data Structure Documentation

### ◆ fsp_pack_version_t

| union fsp_pack_version_t | | |
|---|---|---|
| FSP Pack version structure | | |
| Data Fields | | |
| uint32_t | version_id | Version id |

| | | |
|---|---|---|
| struct version_id_b_s | version_id_b | |

### Macro Definition Documentation

#### ◆ FSP_VERSION_MAJOR

| #define FSP_VERSION_MAJOR |
|---|
| FSP pack major version. |

#### ◆ FSP_VERSION_MINOR

| #define FSP_VERSION_MINOR |
|---|
| FSP pack minor version. |

#### ◆ FSP_VERSION_PATCH

| #define FSP_VERSION_PATCH |
|---|
| FSP pack patch version. |

#### ◆ FSP_VERSION_BUILD

| #define FSP_VERSION_BUILD |
|---|
| FSP pack version build number (currently unused). |

#### ◆ FSP_VERSION_STRING

| #define FSP_VERSION_STRING |
|---|
| Public FSP version name. |

#### ◆ FSP_VERSION_BUILD_STRING

| #define FSP_VERSION_BUILD_STRING |
|---|
| Unique FSP version ID. |

## 5.1.1 BSP I/O access

BSP

### Functions

| | |
|---|---|
| __STATIC_INLINE uint32_t | R_BSP_PinRead (bsp_io_port_pin_t pin) |
| __STATIC_INLINE void | R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level) |
| __STATIC_INLINE void | R_BSP_PinCfg (bsp_io_port_pin_t pin, uint32_t cfg) |
| __STATIC_INLINE void | R_BSP_PinAccessEnable (void) |
| __STATIC_INLINE void | R_BSP_PinAccessDisable (void) |

## Detailed Description

This module provides basic read/write access to port pins.

### Enumerations

| | |
|---|---|
| enum | bsp_io_level_t |
| enum | bsp_io_direction_t |
| enum | bsp_io_port_t |
| enum | bsp_io_port_pin_t |

## Enumeration Type Documentation

### ◆ bsp_io_level_t

| enum bsp_io_level_t | |
|---|---|
| Levels that can be set and read for individual pins | |
| Enumerator | |
| BSP_IO_LEVEL_LOW | Low. |
| BSP_IO_LEVEL_HIGH | High. |

### ◆ bsp_io_direction_t

| enum bsp_io_direction_t | |
|---|---|
| Direction of individual pins | |
| Enumerator | |
| BSP_IO_DIRECTION_INPUT | Input. |
| BSP_IO_DIRECTION_OUTPUT | Output. |

### ◆ bsp_io_port_t

| enum bsp_io_port_t | |
|---|---|
| Superset list of all possible IO ports. | |
| Enumerator | |
| BSP_IO_PORT_00 | IO port 0. |
| BSP_IO_PORT_01 | IO port 1. |
| BSP_IO_PORT_02 | IO port 2. |
| BSP_IO_PORT_03 | IO port 3. |
| BSP_IO_PORT_04 | IO port 4. |
| BSP_IO_PORT_05 | IO port 5. |
| BSP_IO_PORT_06 | IO port 6. |
| BSP_IO_PORT_07 | IO port 7. |
| BSP_IO_PORT_08 | IO port 8. |
| BSP_IO_PORT_09 | IO port 9. |
| BSP_IO_PORT_10 | IO port 10. |
| BSP_IO_PORT_11 | IO port 11. |
| BSP_IO_PORT_12 | IO port 12. |
| BSP_IO_PORT_13 | IO port 13. |
| BSP_IO_PORT_14 | IO port 14. |

### ◆ bsp_io_port_pin_t

| enum bsp_io_port_pin_t | |
|---|---|
| Superset list of all possible IO port pins. | |
| Enumerator | |
| BSP_IO_PORT_00_PIN_00 | IO port 0 pin 0. |
| BSP_IO_PORT_00_PIN_01 | IO port 0 pin 1. |

| | |
|---|---|
| BSP_IO_PORT_00_PIN_02 | IO port 0 pin 2. |
| BSP_IO_PORT_00_PIN_03 | IO port 0 pin 3. |
| BSP_IO_PORT_00_PIN_04 | IO port 0 pin 4. |
| BSP_IO_PORT_00_PIN_05 | IO port 0 pin 5. |
| BSP_IO_PORT_00_PIN_06 | IO port 0 pin 6. |
| BSP_IO_PORT_00_PIN_07 | IO port 0 pin 7. |
| BSP_IO_PORT_00_PIN_08 | IO port 0 pin 8. |
| BSP_IO_PORT_00_PIN_09 | IO port 0 pin 9. |
| BSP_IO_PORT_00_PIN_10 | IO port 0 pin 10. |
| BSP_IO_PORT_00_PIN_11 | IO port 0 pin 11. |
| BSP_IO_PORT_00_PIN_12 | IO port 0 pin 12. |
| BSP_IO_PORT_00_PIN_13 | IO port 0 pin 13. |
| BSP_IO_PORT_00_PIN_14 | IO port 0 pin 14. |
| BSP_IO_PORT_00_PIN_15 | IO port 0 pin 15. |
| BSP_IO_PORT_01_PIN_00 | IO port 1 pin 0. |
| BSP_IO_PORT_01_PIN_01 | IO port 1 pin 1. |
| BSP_IO_PORT_01_PIN_02 | IO port 1 pin 2. |
| BSP_IO_PORT_01_PIN_03 | IO port 1 pin 3. |
| BSP_IO_PORT_01_PIN_04 | IO port 1 pin 4. |
| BSP_IO_PORT_01_PIN_05 | IO port 1 pin 5. |
| BSP_IO_PORT_01_PIN_06 | IO port 1 pin 6. |
| BSP_IO_PORT_01_PIN_07 | IO port 1 pin 7. |
| BSP_IO_PORT_01_PIN_08 | IO port 1 pin 8. |
| BSP_IO_PORT_01_PIN_09 | IO port 1 pin 9. |

| | |
|---|---|
| BSP_IO_PORT_01_PIN_10 | IO port 1 pin 10. |
| BSP_IO_PORT_01_PIN_11 | IO port 1 pin 11. |
| BSP_IO_PORT_01_PIN_12 | IO port 1 pin 12. |
| BSP_IO_PORT_01_PIN_13 | IO port 1 pin 13. |
| BSP_IO_PORT_01_PIN_14 | IO port 1 pin 14. |
| BSP_IO_PORT_01_PIN_15 | IO port 1 pin 15. |
| BSP_IO_PORT_02_PIN_00 | IO port 2 pin 0. |
| BSP_IO_PORT_02_PIN_01 | IO port 2 pin 1. |
| BSP_IO_PORT_02_PIN_02 | IO port 2 pin 2. |
| BSP_IO_PORT_02_PIN_03 | IO port 2 pin 3. |
| BSP_IO_PORT_02_PIN_04 | IO port 2 pin 4. |
| BSP_IO_PORT_02_PIN_05 | IO port 2 pin 5. |
| BSP_IO_PORT_02_PIN_06 | IO port 2 pin 6. |
| BSP_IO_PORT_02_PIN_07 | IO port 2 pin 7. |
| BSP_IO_PORT_02_PIN_08 | IO port 2 pin 8. |
| BSP_IO_PORT_02_PIN_09 | IO port 2 pin 9. |
| BSP_IO_PORT_02_PIN_10 | IO port 2 pin 10. |
| BSP_IO_PORT_02_PIN_11 | IO port 2 pin 11. |
| BSP_IO_PORT_02_PIN_12 | IO port 2 pin 12. |
| BSP_IO_PORT_02_PIN_13 | IO port 2 pin 13. |
| BSP_IO_PORT_02_PIN_14 | IO port 2 pin 14. |
| BSP_IO_PORT_02_PIN_15 | IO port 2 pin 15. |
| BSP_IO_PORT_03_PIN_00 | IO port 3 pin 0. |
| BSP_IO_PORT_03_PIN_01 | IO port 3 pin 1. |

| | |
|---|---|
| BSP_IO_PORT_03_PIN_02 | IO port 3 pin 2. |
| BSP_IO_PORT_03_PIN_03 | IO port 3 pin 3. |
| BSP_IO_PORT_03_PIN_04 | IO port 3 pin 4. |
| BSP_IO_PORT_03_PIN_05 | IO port 3 pin 5. |
| BSP_IO_PORT_03_PIN_06 | IO port 3 pin 6. |
| BSP_IO_PORT_03_PIN_07 | IO port 3 pin 7. |
| BSP_IO_PORT_03_PIN_08 | IO port 3 pin 8. |
| BSP_IO_PORT_03_PIN_09 | IO port 3 pin 9. |
| BSP_IO_PORT_03_PIN_10 | IO port 3 pin 10. |
| BSP_IO_PORT_03_PIN_11 | IO port 3 pin 11. |
| BSP_IO_PORT_03_PIN_12 | IO port 3 pin 12. |
| BSP_IO_PORT_03_PIN_13 | IO port 3 pin 13. |
| BSP_IO_PORT_03_PIN_14 | IO port 3 pin 14. |
| BSP_IO_PORT_03_PIN_15 | IO port 3 pin 15. |
| BSP_IO_PORT_04_PIN_00 | IO port 4 pin 0. |
| BSP_IO_PORT_04_PIN_01 | IO port 4 pin 1. |
| BSP_IO_PORT_04_PIN_02 | IO port 4 pin 2. |
| BSP_IO_PORT_04_PIN_03 | IO port 4 pin 3. |
| BSP_IO_PORT_04_PIN_04 | IO port 4 pin 4. |
| BSP_IO_PORT_04_PIN_05 | IO port 4 pin 5. |
| BSP_IO_PORT_04_PIN_06 | IO port 4 pin 6. |
| BSP_IO_PORT_04_PIN_07 | IO port 4 pin 7. |
| BSP_IO_PORT_04_PIN_08 | IO port 4 pin 8. |
| BSP_IO_PORT_04_PIN_09 | IO port 4 pin 9. |

| | |
|---|---|
| BSP_IO_PORT_04_PIN_10 | IO port 4 pin 10. |
| BSP_IO_PORT_04_PIN_11 | IO port 4 pin 11. |
| BSP_IO_PORT_04_PIN_12 | IO port 4 pin 12. |
| BSP_IO_PORT_04_PIN_13 | IO port 4 pin 13. |
| BSP_IO_PORT_04_PIN_14 | IO port 4 pin 14. |
| BSP_IO_PORT_04_PIN_15 | IO port 4 pin 15. |
| BSP_IO_PORT_05_PIN_00 | IO port 5 pin 0. |
| BSP_IO_PORT_05_PIN_01 | IO port 5 pin 1. |
| BSP_IO_PORT_05_PIN_02 | IO port 5 pin 2. |
| BSP_IO_PORT_05_PIN_03 | IO port 5 pin 3. |
| BSP_IO_PORT_05_PIN_04 | IO port 5 pin 4. |
| BSP_IO_PORT_05_PIN_05 | IO port 5 pin 5. |
| BSP_IO_PORT_05_PIN_06 | IO port 5 pin 6. |
| BSP_IO_PORT_05_PIN_07 | IO port 5 pin 7. |
| BSP_IO_PORT_05_PIN_08 | IO port 5 pin 8. |
| BSP_IO_PORT_05_PIN_09 | IO port 5 pin 9. |
| BSP_IO_PORT_05_PIN_10 | IO port 5 pin 10. |
| BSP_IO_PORT_05_PIN_11 | IO port 5 pin 11. |
| BSP_IO_PORT_05_PIN_12 | IO port 5 pin 12. |
| BSP_IO_PORT_05_PIN_13 | IO port 5 pin 13. |
| BSP_IO_PORT_05_PIN_14 | IO port 5 pin 14. |
| BSP_IO_PORT_05_PIN_15 | IO port 5 pin 15. |
| BSP_IO_PORT_06_PIN_00 | IO port 6 pin 0. |
| BSP_IO_PORT_06_PIN_01 | IO port 6 pin 1. |

| | |
|---|---|
| BSP_IO_PORT_06_PIN_02 | IO port 6 pin 2. |
| BSP_IO_PORT_06_PIN_03 | IO port 6 pin 3. |
| BSP_IO_PORT_06_PIN_04 | IO port 6 pin 4. |
| BSP_IO_PORT_06_PIN_05 | IO port 6 pin 5. |
| BSP_IO_PORT_06_PIN_06 | IO port 6 pin 6. |
| BSP_IO_PORT_06_PIN_07 | IO port 6 pin 7. |
| BSP_IO_PORT_06_PIN_08 | IO port 6 pin 8. |
| BSP_IO_PORT_06_PIN_09 | IO port 6 pin 9. |
| BSP_IO_PORT_06_PIN_10 | IO port 6 pin 10. |
| BSP_IO_PORT_06_PIN_11 | IO port 6 pin 11. |
| BSP_IO_PORT_06_PIN_12 | IO port 6 pin 12. |
| BSP_IO_PORT_06_PIN_13 | IO port 6 pin 13. |
| BSP_IO_PORT_06_PIN_14 | IO port 6 pin 14. |
| BSP_IO_PORT_06_PIN_15 | IO port 6 pin 15. |
| BSP_IO_PORT_07_PIN_00 | IO port 7 pin 0. |
| BSP_IO_PORT_07_PIN_01 | IO port 7 pin 1. |
| BSP_IO_PORT_07_PIN_02 | IO port 7 pin 2. |
| BSP_IO_PORT_07_PIN_03 | IO port 7 pin 3. |
| BSP_IO_PORT_07_PIN_04 | IO port 7 pin 4. |
| BSP_IO_PORT_07_PIN_05 | IO port 7 pin 5. |
| BSP_IO_PORT_07_PIN_06 | IO port 7 pin 6. |
| BSP_IO_PORT_07_PIN_07 | IO port 7 pin 7. |
| BSP_IO_PORT_07_PIN_08 | IO port 7 pin 8. |
| BSP_IO_PORT_07_PIN_09 | IO port 7 pin 9. |

| | |
|---|---|
| BSP_IO_PORT_07_PIN_10 | IO port 7 pin 10. |
| BSP_IO_PORT_07_PIN_11 | IO port 7 pin 11. |
| BSP_IO_PORT_07_PIN_12 | IO port 7 pin 12. |
| BSP_IO_PORT_07_PIN_13 | IO port 7 pin 13. |
| BSP_IO_PORT_07_PIN_14 | IO port 7 pin 14. |
| BSP_IO_PORT_07_PIN_15 | IO port 7 pin 15. |
| BSP_IO_PORT_08_PIN_00 | IO port 8 pin 0. |
| BSP_IO_PORT_08_PIN_01 | IO port 8 pin 1. |
| BSP_IO_PORT_08_PIN_02 | IO port 8 pin 2. |
| BSP_IO_PORT_08_PIN_03 | IO port 8 pin 3. |
| BSP_IO_PORT_08_PIN_04 | IO port 8 pin 4. |
| BSP_IO_PORT_08_PIN_05 | IO port 8 pin 5. |
| BSP_IO_PORT_08_PIN_06 | IO port 8 pin 6. |
| BSP_IO_PORT_08_PIN_07 | IO port 8 pin 7. |
| BSP_IO_PORT_08_PIN_08 | IO port 8 pin 8. |
| BSP_IO_PORT_08_PIN_09 | IO port 8 pin 9. |
| BSP_IO_PORT_08_PIN_10 | IO port 8 pin 10. |
| BSP_IO_PORT_08_PIN_11 | IO port 8 pin 11. |
| BSP_IO_PORT_08_PIN_12 | IO port 8 pin 12. |
| BSP_IO_PORT_08_PIN_13 | IO port 8 pin 13. |
| BSP_IO_PORT_08_PIN_14 | IO port 8 pin 14. |
| BSP_IO_PORT_08_PIN_15 | IO port 8 pin 15. |
| BSP_IO_PORT_09_PIN_00 | IO port 9 pin 0. |
| BSP_IO_PORT_09_PIN_01 | IO port 9 pin 1. |

| | |
|---|---|
| BSP_IO_PORT_09_PIN_02 | IO port 9 pin 2. |
| BSP_IO_PORT_09_PIN_03 | IO port 9 pin 3. |
| BSP_IO_PORT_09_PIN_04 | IO port 9 pin 4. |
| BSP_IO_PORT_09_PIN_05 | IO port 9 pin 5. |
| BSP_IO_PORT_09_PIN_06 | IO port 9 pin 6. |
| BSP_IO_PORT_09_PIN_07 | IO port 9 pin 7. |
| BSP_IO_PORT_09_PIN_08 | IO port 9 pin 8. |
| BSP_IO_PORT_09_PIN_09 | IO port 9 pin 9. |
| BSP_IO_PORT_09_PIN_10 | IO port 9 pin 10. |
| BSP_IO_PORT_09_PIN_11 | IO port 9 pin 11. |
| BSP_IO_PORT_09_PIN_12 | IO port 9 pin 12. |
| BSP_IO_PORT_09_PIN_13 | IO port 9 pin 13. |
| BSP_IO_PORT_09_PIN_14 | IO port 9 pin 14. |
| BSP_IO_PORT_09_PIN_15 | IO port 9 pin 15. |
| BSP_IO_PORT_10_PIN_00 | IO port 10 pin 0. |
| BSP_IO_PORT_10_PIN_01 | IO port 10 pin 1. |
| BSP_IO_PORT_10_PIN_02 | IO port 10 pin 2. |
| BSP_IO_PORT_10_PIN_03 | IO port 10 pin 3. |
| BSP_IO_PORT_10_PIN_04 | IO port 10 pin 4. |
| BSP_IO_PORT_10_PIN_05 | IO port 10 pin 5. |
| BSP_IO_PORT_10_PIN_06 | IO port 10 pin 6. |
| BSP_IO_PORT_10_PIN_07 | IO port 10 pin 7. |
| BSP_IO_PORT_10_PIN_08 | IO port 10 pin 8. |
| BSP_IO_PORT_10_PIN_09 | IO port 10 pin 9. |

| | |
|---|---|
| BSP_IO_PORT_10_PIN_10 | IO port 10 pin 10. |
| BSP_IO_PORT_10_PIN_11 | IO port 10 pin 11. |
| BSP_IO_PORT_10_PIN_12 | IO port 10 pin 12. |
| BSP_IO_PORT_10_PIN_13 | IO port 10 pin 13. |
| BSP_IO_PORT_10_PIN_14 | IO port 10 pin 14. |
| BSP_IO_PORT_10_PIN_15 | IO port 10 pin 15. |
| BSP_IO_PORT_11_PIN_00 | IO port 11 pin 0. |
| BSP_IO_PORT_11_PIN_01 | IO port 11 pin 1. |
| BSP_IO_PORT_11_PIN_02 | IO port 11 pin 2. |
| BSP_IO_PORT_11_PIN_03 | IO port 11 pin 3. |
| BSP_IO_PORT_11_PIN_04 | IO port 11 pin 4. |
| BSP_IO_PORT_11_PIN_05 | IO port 11 pin 5. |
| BSP_IO_PORT_11_PIN_06 | IO port 11 pin 6. |
| BSP_IO_PORT_11_PIN_07 | IO port 11 pin 7. |
| BSP_IO_PORT_11_PIN_08 | IO port 11 pin 8. |
| BSP_IO_PORT_11_PIN_09 | IO port 11 pin 9. |
| BSP_IO_PORT_11_PIN_10 | IO port 11 pin 10. |
| BSP_IO_PORT_11_PIN_11 | IO port 11 pin 11. |
| BSP_IO_PORT_11_PIN_12 | IO port 11 pin 12. |
| BSP_IO_PORT_11_PIN_13 | IO port 11 pin 13. |
| BSP_IO_PORT_11_PIN_14 | IO port 11 pin 14. |
| BSP_IO_PORT_11_PIN_15 | IO port 11 pin 15. |
| BSP_IO_PORT_12_PIN_00 | IO port 12 pin 0. |
| BSP_IO_PORT_12_PIN_01 | IO port 12 pin 1. |

| | |
|---|---|
| BSP_IO_PORT_12_PIN_02 | IO port 12 pin 2. |
| BSP_IO_PORT_12_PIN_03 | IO port 12 pin 3. |
| BSP_IO_PORT_12_PIN_04 | IO port 12 pin 4. |
| BSP_IO_PORT_12_PIN_05 | IO port 12 pin 5. |
| BSP_IO_PORT_12_PIN_06 | IO port 12 pin 6. |
| BSP_IO_PORT_12_PIN_07 | IO port 12 pin 7. |
| BSP_IO_PORT_12_PIN_08 | IO port 12 pin 8. |
| BSP_IO_PORT_12_PIN_09 | IO port 12 pin 9. |
| BSP_IO_PORT_12_PIN_10 | IO port 12 pin 10. |
| BSP_IO_PORT_12_PIN_11 | IO port 12 pin 11. |
| BSP_IO_PORT_12_PIN_12 | IO port 12 pin 12. |
| BSP_IO_PORT_12_PIN_13 | IO port 12 pin 13. |
| BSP_IO_PORT_12_PIN_14 | IO port 12 pin 14. |
| BSP_IO_PORT_12_PIN_15 | IO port 12 pin 15. |
| BSP_IO_PORT_13_PIN_00 | IO port 13 pin 0. |
| BSP_IO_PORT_13_PIN_01 | IO port 13 pin 1. |
| BSP_IO_PORT_13_PIN_02 | IO port 13 pin 2. |
| BSP_IO_PORT_13_PIN_03 | IO port 13 pin 3. |
| BSP_IO_PORT_13_PIN_04 | IO port 13 pin 4. |
| BSP_IO_PORT_13_PIN_05 | IO port 13 pin 5. |
| BSP_IO_PORT_13_PIN_06 | IO port 13 pin 6. |
| BSP_IO_PORT_13_PIN_07 | IO port 13 pin 7. |
| BSP_IO_PORT_13_PIN_08 | IO port 13 pin 8. |
| BSP_IO_PORT_13_PIN_09 | IO port 13 pin 9. |

| | |
|---|---|
| BSP_IO_PORT_13_PIN_10 | IO port 13 pin 10. |
| BSP_IO_PORT_13_PIN_11 | IO port 13 pin 11. |
| BSP_IO_PORT_13_PIN_12 | IO port 13 pin 12. |
| BSP_IO_PORT_13_PIN_13 | IO port 13 pin 13. |
| BSP_IO_PORT_13_PIN_14 | IO port 13 pin 14. |
| BSP_IO_PORT_13_PIN_15 | IO port 13 pin 15. |
| BSP_IO_PORT_14_PIN_00 | IO port 14 pin 0. |
| BSP_IO_PORT_14_PIN_01 | IO port 14 pin 1. |
| BSP_IO_PORT_14_PIN_02 | IO port 14 pin 2. |
| BSP_IO_PORT_14_PIN_03 | IO port 14 pin 3. |
| BSP_IO_PORT_14_PIN_04 | IO port 14 pin 4. |
| BSP_IO_PORT_14_PIN_05 | IO port 14 pin 5. |
| BSP_IO_PORT_14_PIN_06 | IO port 14 pin 6. |
| BSP_IO_PORT_14_PIN_07 | IO port 14 pin 7. |
| BSP_IO_PORT_14_PIN_08 | IO port 14 pin 8. |
| BSP_IO_PORT_14_PIN_09 | IO port 14 pin 9. |
| BSP_IO_PORT_14_PIN_10 | IO port 14 pin 10. |
| BSP_IO_PORT_14_PIN_11 | IO port 14 pin 11. |
| BSP_IO_PORT_14_PIN_12 | IO port 14 pin 12. |
| BSP_IO_PORT_14_PIN_13 | IO port 14 pin 13. |
| BSP_IO_PORT_14_PIN_14 | IO port 14 pin 14. |
| BSP_IO_PORT_14_PIN_15 | IO port 14 pin 15. |
| BSP_IO_PORT_FF_PIN_FF | Invalid IO port. |

## Function Documentation

### ◆ R_BSP_PinRead()

__STATIC_INLINE uint32_t R_BSP_PinRead ( bsp_io_port_pin_t *pin*)

Read the current input level of the pin.

**Parameters**

| | | |
|---|---|---|
| [in] | pin | The pin |

**Return values**

| | |
|---|---|
| Current | input level |

### ◆ R_BSP_PinWrite()

__STATIC_INLINE void R_BSP_PinWrite ( bsp_io_port_pin_t *pin*, bsp_io_level_t *level* )

Set a pin to output and set the output level to the level provided. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

**Parameters**

| | | |
|---|---|---|
| [in] | pin | The pin |
| [in] | level | The level |

### ◆ R_BSP_PinCfg()

__STATIC_INLINE void R_BSP_PinCfg ( bsp_io_port_pin_t *pin*, uint32_t *cfg* )

Configure a pin. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

**Parameters**

| | | |
|---|---|---|
| [in] | pin | The pin |
| [in] | cfg | Configuration for the pin (PmnPFS register setting) |

### ◆ R_BSP_PinAccessEnable()

__STATIC_INLINE void R_BSP_PinAccessEnable ( void )

Enable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

### ◆ R_BSP_PinAccessDisable()

| __STATIC_INLINE void R_BSP_PinAccessDisable ( void ) |
| --- |
| Disable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code. |

## 5.1.2 BSP OSPI_B support

BSP

Code that initializes the OSPI_B device memory.

## 5.1.3 BSP SDRAM support

BSP

### Functions

| | |
| --- | --- |
| void | R_BSP_SdramInit (bool init_memory)<br>Initializes SDRAM. More... |
| void | R_BSP_SdramSelfRefreshEnable (void)<br>Changes SDRAM from Auto-refresh to Self-refresh. More... |
| void | R_BSP_SdramSelfRefreshDisable (void)<br>Changes SDRAM from Self-refresh to Auto-refresh. More... |

### Detailed Description

Code that initializes the SDRAMC and SDR SDRAM device memory.

### Function Documentation

### ◆ R_BSP_SdramInit()

| void R_BSP_SdramInit ( bool *init_memory*) |
|---|
| Initializes SDRAM.<br><br>**Parameters**<br>init_memory — If true, this function will execute the initialization of the external modules. Otherwise, it will only initialize the SDRAMC and leave the memory in self-refresh mode.<br><br>This function initializes SDRAMC and SDR SDRAM device.<br><br>***Note***<br>*This function must only be called once after reset.* |

### ◆ R_BSP_SdramSelfRefreshEnable()

| void R_BSP_SdramSelfRefreshEnable ( void ) |
|---|
| Changes SDRAM from Auto-refresh to Self-refresh.<br><br>This function allows Software Standby and Deep Software Standby modes to be entered without data loss.<br><br>***Note***<br>*SDRAM cannot be accessed after calling this function. Use R_BSP_SdramSelfRefreshDisable to resume normal SDRAM operation.* |

### ◆ R_BSP_SdramSelfRefreshDisable()

| void R_BSP_SdramSelfRefreshDisable ( void ) |
|---|
| Changes SDRAM from Self-refresh to Auto-refresh.<br><br>This function changes back to Auto-refresh and allows normal SDRAM operation to resume. |

## 5.1.4 Common Error Codes

BSP

### Detailed Description

All FSP modules share these common error codes.

### Macros

| | |
|---|---|
| #define | FSP_PARAMETER_NOT_USED(p) |
| #define | FSP_CPP_HEADER |
| #define | FSP_HEADER |
| #define | FSP_SECURE_ARGUMENT |

**Enumerations**

| | |
|---|---|
| enum | fsp_err_t |

## Macro Definition Documentation

### ◆ FSP_PARAMETER_NOT_USED

```
#define FSP_PARAMETER_NOT_USED ( p)
```

This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.

### ◆ FSP_CPP_HEADER

```
#define FSP_CPP_HEADER
```

Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.

### ◆ FSP_HEADER

```
#define FSP_HEADER
```

FSP Header and Footer definitions

### ◆ FSP_SECURE_ARGUMENT

```
#define FSP_SECURE_ARGUMENT
```

Macro to be used when argument to function is ignored since function call is NSC and the parameter is statically defined on the Secure side.

## Enumeration Type Documentation

### ◆ fsp_err_t

| enum fsp_err_t | |
|---|---|
| Common error codes | |
| Enumerator | |
| FSP_ERR_ASSERTION | A critical assertion has failed. |
| FSP_ERR_INVALID_POINTER | Pointer points to invalid memory location. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |
| FSP_ERR_INVALID_CHANNEL | Selected channel does not exist. |
| FSP_ERR_INVALID_MODE | Unsupported or incorrect mode. |
| FSP_ERR_UNSUPPORTED | Selected mode not supported by this API. |
| FSP_ERR_NOT_OPEN | Requested channel is not configured or API not open. |
| FSP_ERR_IN_USE | Channel/peripheral is running/busy. |
| FSP_ERR_OUT_OF_MEMORY | Allocate more memory in the driver's cfg.h. |
| FSP_ERR_HW_LOCKED | Hardware is locked. |
| FSP_ERR_IRQ_BSP_DISABLED | IRQ not enabled in BSP. |
| FSP_ERR_OVERFLOW | Hardware overflow. |
| FSP_ERR_UNDERFLOW | Hardware underflow. |
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_APPROXIMATION | Could not set value to exact result. |
| FSP_ERR_CLAMPED | Value had to be limited for some reason. |
| FSP_ERR_INVALID_RATE | Selected rate could not be met. |
| FSP_ERR_ABORTED | An operation was aborted. |
| FSP_ERR_NOT_ENABLED | Requested operation is not enabled. |
| FSP_ERR_TIMEOUT | Timeout error. |

| | |
|---|---|
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks supplied. |
| FSP_ERR_INVALID_ADDRESS | Invalid address supplied. |
| FSP_ERR_INVALID_SIZE | Invalid size/length supplied for operation. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |
| FSP_ERR_ERASE_FAILED | Erase operation failed. |
| FSP_ERR_INVALID_CALL | Invalid function call is made. |
| FSP_ERR_INVALID_HW_CONDITION | Detected hardware is in invalid condition. |
| FSP_ERR_INVALID_FACTORY_FLASH | Factory flash is not available on this MCU. |
| FSP_ERR_INVALID_STATE | API or command not valid in the current state. |
| FSP_ERR_NOT_ERASED | Erase verification failed. |
| FSP_ERR_SECTOR_RELEASE_FAILED | Sector release failed. |
| FSP_ERR_NOT_INITIALIZED | Required initialization not complete. |
| FSP_ERR_NOT_FOUND | The requested item could not be found. |
| FSP_ERR_NO_CALLBACK_MEMORY | Non-secure callback memory not provided for non-secure callback. |
| FSP_ERR_BUFFER_EMPTY | No data available in buffer. |
| FSP_ERR_INVALID_DATA | Accuracy of data is not guaranteed. |
| FSP_ERR_INTERNAL | Internal error. |
| FSP_ERR_WAIT_ABORTED | Wait aborted. |
| FSP_ERR_FRAMING | Framing error occurs. |
| FSP_ERR_BREAK_DETECT | Break signal detects. |
| FSP_ERR_PARITY | Parity error occurs. |
| FSP_ERR_RXBUF_OVERFLOW | Receive queue overflow. |
| FSP_ERR_QUEUE_UNAVAILABLE | Can't open s/w queue. |
| FSP_ERR_INSUFFICIENT_SPACE | Not enough space in transmission circular |

| | |
|---|---|
| | buffer. |
| FSP_ERR_INSUFFICIENT_DATA | Not enough data in receive circular buffer. |
| FSP_ERR_TRANSFER_ABORTED | The data transfer was aborted. |
| FSP_ERR_MODE_FAULT | Mode fault error. |
| FSP_ERR_READ_OVERFLOW | Read overflow. |
| FSP_ERR_SPI_PARITY | Parity error. |
| FSP_ERR_OVERRUN | Overrun error. |
| FSP_ERR_CLOCK_INACTIVE | Inactive clock specified as system clock. |
| FSP_ERR_CLOCK_ACTIVE | Active clock source cannot be modified without stopping first. |
| FSP_ERR_NOT_STABILIZED | Clock has not stabilized after its been turned on/off. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL initialization attempted when PLL source is turned off. |
| FSP_ERR_OSC_STOP_DET_ENABLED | Illegal attempt to stop LOCO when Oscillation stop is enabled. |
| FSP_ERR_OSC_STOP_DETECTED | The Oscillation stop detection status flag is set. |
| FSP_ERR_OSC_STOP_CLOCK_ACTIVE | Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active. |
| FSP_ERR_CLKOUT_EXCEEDED | Output on target output clock pin exceeds maximum supported limit. |
| FSP_ERR_USB_MODULE_ENABLED | USB clock configure request with USB Module enabled. |
| FSP_ERR_HARDWARE_TIMEOUT | A register read or write timed out. |
| FSP_ERR_LOW_VOLTAGE_MODE | Invalid clock setting attempted in low voltage mode. |
| FSP_ERR_PE_FAILURE | Unable to enter Programming mode. |
| FSP_ERR_CMD_LOCKED | Peripheral in command locked state. |
| FSP_ERR_FCLK | |

| | |
|---|---|
| | FCLK must be >= 4 MHz. |
| FSP_ERR_INVALID_LINKED_ADDRESS | Function or data are linked at an invalid region of memory. |
| FSP_ERR_BLANK_CHECK_FAILED | Blank check operation failed. |
| FSP_ERR_HUK_ZEROIZATION | W-HUK zeroization is in progress. |
| FSP_ERR_INVALID_CAC_REF_CLOCK | Measured clock rate < reference clock rate. |
| FSP_ERR_INVALID_RESULT | The result of one or more calculations was +/- infinity. |
| FSP_ERR_CLOCK_GENERATION | Clock cannot be specified as system clock. |
| FSP_ERR_INVALID_TIMING_SETTING | Invalid timing parameter. |
| FSP_ERR_INVALID_LAYER_SETTING | Invalid layer parameter. |
| FSP_ERR_INVALID_ALIGNMENT | Invalid memory alignment found. |
| FSP_ERR_INVALID_GAMMA_SETTING | Invalid gamma correction parameter. |
| FSP_ERR_INVALID_LAYER_FORMAT | Invalid color format in layer. |
| FSP_ERR_INVALID_UPDATE_TIMING | Invalid timing for register update. |
| FSP_ERR_INVALID_CLUT_ACCESS | Invalid access to CLUT entry. |
| FSP_ERR_INVALID_FADE_SETTING | Invalid fade-in/fade-out setting. |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid gamma correction parameter. |
| FSP_ERR_JPEG_ERR | JPEG error. |
| FSP_ERR_JPEG_SOI_NOT_DETECTED | SOI not detected until EOI detected. |
| FSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED | SOF1 to SOFF detected. |
| FSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT | Unprovided pixel format detected. |
| FSP_ERR_JPEG_SOF_ACCURACY_ERROR | SOF accuracy error: other than 8 detected. |
| FSP_ERR_JPEG_DQT_ACCURACY_ERROR | DQT accuracy error: other than 0 detected. |
| FSP_ERR_JPEG_COMPONENT_ERROR1 | Component error 1: the number of SOF0 header components detected is other than 1, 3, or 4. |

| | |
|---|---|
| FSP_ERR_JPEG_COMPONENT_ERROR2 | Component error 2: the number of components differs between SOF0 header and SOS. |
| FSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED | SOF0, DQT, and DHT not detected when SOS detected. |
| FSP_ERR_JPEG_SOS_NOT_DETECTED | SOS not detected: SOS not detected until EOI detected. |
| FSP_ERR_JPEG_EOI_NOT_DETECTED | EOI not detected (default) |
| FSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR | Restart interval data number error detected. |
| FSP_ERR_JPEG_IMAGE_SIZE_ERROR | Image size error detected. |
| FSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR | Last MCU data number error detected. |
| FSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR | Block data number error detected. |
| FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH | User provided buffer size not enough. |
| FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE | JPEG Image size is not aligned with MCU. |
| FSP_ERR_CALIBRATE_FAILED | Calibration failed. |
| FSP_ERR_IIRFA_ECC_1BIT | 1-bit ECC error detected |
| FSP_ERR_IIRFA_ECC_2BIT | 2-bit ECC error detected |
| FSP_ERR_IP_HARDWARE_NOT_PRESENT | Requested IP does not exist on this device. |
| FSP_ERR_IP_UNIT_NOT_PRESENT | Requested unit does not exist on this device. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel does not exist on this device. |
| FSP_ERR_NO_MORE_BUFFER | No more buffer found in the memory block pool. |
| FSP_ERR_ILLEGAL_BUFFER_ADDRESS | Buffer address is out of block memory pool. |
| FSP_ERR_INVALID_WORKBUFFER_SIZE | Work buffer size is invalid. |
| FSP_ERR_INVALID_MSG_BUFFER_SIZE | Message buffer size is invalid. |
| FSP_ERR_TOO_MANY_BUFFERS | Number of buffer is too many. |
| FSP_ERR_NO_SUBSCRIBER_FOUND | No message subscriber found. |

| | |
|---|---|
| FSP_ERR_MESSAGE_QUEUE_EMPTY | No message found in the message queue. |
| FSP_ERR_MESSAGE_QUEUE_FULL | No room for new message in the message queue. |
| FSP_ERR_ILLEGAL_SUBSCRIBER_LISTS | Message subscriber lists is illegal. |
| FSP_ERR_BUFFER_RELEASED | Buffer has been released. |
| FSP_ERR_D2D_ERROR_INIT | D/AVE 2D has an error in the initialization. |
| FSP_ERR_D2D_ERROR_DEINIT | D/AVE 2D has an error in the initialization. |
| FSP_ERR_D2D_ERROR_RENDERING | D/AVE 2D has an error in the rendering. |
| FSP_ERR_D2D_ERROR_SIZE | D/AVE 2D has an error in the rendering. |
| FSP_ERR_ETHER_ERROR_NO_DATA | No Data in Receive buffer. |
| FSP_ERR_ETHER_ERROR_LINK | ETHERC/EDMAC has an error in the Auto-negotiation. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, and transmission/reception is not enabled. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | Transmit buffer is not empty. |
| FSP_ERR_ETHER_ERROR_FILTERING | Detect multicast frame when multicast frame filtering enable. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | ETHERC/EDMAC has an error in the phy communication. |
| FSP_ERR_ETHER_RECEIVE_BUFFER_ACTIVE | Receive buffer is active. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | PHY has an error in the Auto-negotiation. |
| FSP_ERR_QUEUE_FULL | Queue is full, cannot queue another data. |
| FSP_ERR_QUEUE_EMPTY | Queue is empty, no data to dequeue. |
| FSP_ERR_CTSU_SCANNING | Scanning. |
| FSP_ERR_CTSU_NOT_GET_DATA | Not processed previous scan data. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

| | |
|---|---|
| FSP_ERR_CTSU_DIAG_NOT_YET | Diagnosis of data collected no yet. |
| FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE | Diagnosis of LDO over voltage failed. |
| FSP_ERR_CTSU_DIAG_CCO_HIGH | Diagnosis of CCO into 19.2uA failed. |
| FSP_ERR_CTSU_DIAG_CCO_LOW | Diagnosis of CCO into 2.4uA failed. |
| FSP_ERR_CTSU_DIAG_SSCG | Diagnosis of SSCG frequency failed. |
| FSP_ERR_CTSU_DIAG_DAC | Diagnosis of non-touch count value failed. |
| FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE | Diagnosis of LDO output voltage failed. |
| FSP_ERR_CTSU_DIAG_OVER_VOLTAGE | Diagnosis of over voltage detection circuit failed. |
| FSP_ERR_CTSU_DIAG_OVER_CURRENT | Diagnosis of over current detection circuit failed. |
| FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE | Diagnosis of LDO internal resistance value failed. |
| FSP_ERR_CTSU_DIAG_CURRENT_SOURCE | Diagnosis of Current source value failed. |
| FSP_ERR_CTSU_DIAG_SENSCLK_GAIN | Diagnosis of SENSCLK frequency gain failed. |
| FSP_ERR_CTSU_DIAG_SUCLK_GAIN | Diagnosis of SUCLK frequency gain failed. |
| FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY | Diagnosis of SUCLK clock recovery function failed. |
| FSP_ERR_CTSU_DIAG_CFC_GAIN | Diagnosis of CFC oscillator gain failed. |
| FSP_ERR_CARD_INIT_FAILED | SD card or eMMC device failed to initialize. |
| FSP_ERR_CARD_NOT_INSERTED | SD card not installed. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low or another operation is ongoing. |
| FSP_ERR_CARD_NOT_INITIALIZED | SD card was removed. |
| FSP_ERR_CARD_WRITE_PROTECTED | Media is write protected. |
| FSP_ERR_TRANSFER_BUSY | Transfer in progress. |
| FSP_ERR_RESPONSE | Card did not respond or responded with an error. |

| | |
|---|---|
| FSP_ERR_MEDIA_FORMAT_FAILED | Media format failed. |
| FSP_ERR_MEDIA_OPEN_FAILED | Media open failed. |
| FSP_ERR_CAN_DATA_UNAVAILABLE | No data available. |
| FSP_ERR_CAN_MODE_SWITCH_FAILED | Switching operation modes failed. |
| FSP_ERR_CAN_INIT_FAILED | Hardware initialization failed. |
| FSP_ERR_CAN_TRANSMIT_NOT_READY | Transmit in progress. |
| FSP_ERR_CAN_RECEIVE_MAILBOX | Mailbox is setup as a receive mailbox. |
| FSP_ERR_CAN_TRANSMIT_MAILBOX | Mailbox is setup as a transmit mailbox. |
| FSP_ERR_CAN_MESSAGE_LOST | Receive message has been overwritten or overrun. |
| FSP_ERR_CAN_TRANSMIT_FIFO_FULL | Transmit FIFO is full. |
| FSP_ERR_WIFI_CONFIG_FAILED | WiFi module Configuration failed. |
| FSP_ERR_WIFI_INIT_FAILED | WiFi module initialization failed. |
| FSP_ERR_WIFI_TRANSMIT_FAILED | Transmission failed. |
| FSP_ERR_WIFI_INVALID_MODE | API called when provisioned in client mode. |
| FSP_ERR_WIFI_FAILED | WiFi Failed. |
| FSP_ERR_WIFI_SCAN_COMPLETE | Wifi scan has completed. |
| FSP_ERR_WIFI_AP_NOT_CONNECTED | WiFi module is not connected to access point. |
| FSP_ERR_WIFI_UNKNOWN_AT_CMD | DA16XXX Unknown AT command Error. |
| FSP_ERR_WIFI_INSUF_PARAM | DA16XXX Insufficient parameter. |
| FSP_ERR_WIFI_TOO_MANY_PARAMS | DA16XXX Too many parameters. |
| FSP_ERR_WIFI_INV_PARAM_VAL | DA16XXX Wrong parameter value. |
| FSP_ERR_WIFI_NO_RESULT | DA16XXX No result. |
| FSP_ERR_WIFI_RSP_BUF_OVFLW | DA16XXX Response buffer overflow. |
| FSP_ERR_WIFI_FUNC_NOT_CONFIG | DA16XXX Function is not configured. |

| | |
|---|---|
| FSP_ERR_WIFI_NVRAM_WR_FAIL | DA16XXX NVRAM write failure. |
| FSP_ERR_WIFI_RET_MEM_WR_FAIL | DA16XXX Retention memory write failure. |
| FSP_ERR_WIFI_UNKNOWN_ERR | DA16XXX unknown error. |
| FSP_ERR_CELLULAR_CONFIG_FAILED | Cellular module Configuration failed. |
| FSP_ERR_CELLULAR_INIT_FAILED | Cellular module initialization failed. |
| FSP_ERR_CELLULAR_TRANSMIT_FAILED | Transmission failed. |
| FSP_ERR_CELLULAR_FW_UPTODATE | Firmware is uptodate. |
| FSP_ERR_CELLULAR_FW_UPGRADE_FAILED | Firmware upgrade failed. |
| FSP_ERR_CELLULAR_FAILED | Cellular Failed. |
| FSP_ERR_CELLULAR_INVALID_STATE | API Called in invalid state. |
| FSP_ERR_CELLULAR_REGISTRATION_FAILED | Cellular Network registration failed. |
| FSP_ERR_BLE_FAILED | BLE operation failed. |
| FSP_ERR_BLE_INIT_FAILED | BLE device initialization failed. |
| FSP_ERR_BLE_CONFIG_FAILED | BLE device configuration failed. |
| FSP_ERR_BLE_PRF_ALREADY_ENABLED | BLE device Profile already enabled. |
| FSP_ERR_BLE_PRF_NOT_ENABLED | BLE device not enabled. |
| FSP_ERR_BLE_ABS_INVALID_OPERATION | Invalid operation is executed. |
| FSP_ERR_BLE_ABS_NOT_FOUND | Valid data or free space is not found. |
| FSP_ERR_CRYPTO_CONTINUE | Continue executing function. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Hardware resource busy. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal I/O buffer is not empty. |
| FSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX | Invalid index. |
| FSP_ERR_CRYPTO_SCE_RETRY | Retry. |
| FSP_ERR_CRYPTO_SCE_VERIFY_FAIL | Verify is failed. |

| | |
|---|---|
| FSP_ERR_CRYPTO_SCE_ALREADY_OPEN | HW SCE module is already opened. |
| FSP_ERR_CRYPTO_NOT_OPEN | Hardware module is not initialized. |
| FSP_ERR_CRYPTO_UNKNOWN | Some unknown error occurred. |
| FSP_ERR_CRYPTO_NULL_POINTER | Null pointer input as a parameter. |
| FSP_ERR_CRYPTO_NOT_IMPLEMENTED | Algorithm/size not implemented. |
| FSP_ERR_CRYPTO_RNG_INVALID_PARAM | An invalid parameter is specified. |
| FSP_ERR_CRYPTO_RNG_FATAL_ERROR | A fatal error occurred. |
| FSP_ERR_CRYPTO_INVALID_SIZE | Size specified is invalid. |
| FSP_ERR_CRYPTO_INVALID_STATE | Function used in an valid state. |
| FSP_ERR_CRYPTO_ALREADY_OPEN | control block is already opened |
| FSP_ERR_CRYPTO_INSTALL_KEY_FAILED | Specified input key is invalid. |
| FSP_ERR_CRYPTO_AUTHENTICATION_FAILED | Authentication failed. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Failure to Init Cipher. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input date is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function call occurred. |
| FSP_ERR_CRYPTO_SCE_LBIST_CHECK_BUSY | LBIST Check BUSY. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | Hardware resource is busy. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Hardware fatal error or unexpected return. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key type is illegal. |
| FSP_ERR_CRYPTO_RSIP_AUTHENTICATION | Authentication failed. |
| FSP_ERR_CRYPTO_RSIP_LBIST_CHECK_BUSY | LBIST Check BUSY. |
| FSP_ERR_CRYPTO_COMMON_NOT_OPENED | Crypto Framework Common is not opened. |

| | |
|---|---|
| FSP_ERR_CRYPTO_HAL_ERROR | Cryoto HAL module returned an error. |
| FSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH | Key buffer size is not enough to generate a key. |
| FSP_ERR_CRYPTO_BUF_OVERFLOW | Attempt to write data larger than what the buffer can hold. |
| FSP_ERR_CRYPTO_INVALID_OPERATION_MODE | Invalid operation mode. |
| FSP_ERR_MESSAGE_TOO_LONG | Message for RSA encryption is too long. |
| FSP_ERR_RSA_DECRYPTION_ERROR | RSA Decryption error. |
| FSP_ERR_SENSOR_INVALID_DATA | Data is invalid.<br>*Note*<br>*SF_CRYPTO APIs may return an error code starting from 0x10000 which is of Crypto module. Refer to sf_cryoto_err.h for Crypto error codes.* |
| FSP_ERR_SENSOR_IN_STABILIZATION | Sensor is stabilizing. |
| FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED | Measurement is not finished. |
| FSP_ERR_COMMS_BUS_NOT_OPEN | Bus is not open. |

### 5.1.5 MCU Board Support Package

BSP

**Functions**

| | |
|---|---|
| fsp_err_t | R_FSP_VersionGet (fsp_pack_version_t *const p_version) |
| uint32_t | R_BSP_SourceClockHzGet (fsp_priv_source_clock_t clock) |
| __STATIC_INLINE IRQn_Type | R_FSP_CurrentIrqGet (void) |
| __STATIC_INLINE uint32_t | R_FSP_SystemClockHzGet (fsp_priv_clock_t clock) |
| __STATIC_INLINE uint32_t | R_FSP_ClockDividerGet (uint32_t ckdivcr) |
| __STATIC_INLINE bsp_unique_id_t const * | R_BSP_UniqueIdGet (void) |

| | |
|---|---|
| __STATIC_INLINE fsp_err_t | R_BSP_PartNumberGet (bsp_part_number_t *const p_part_number) |
| __STATIC_INLINE void | R_BSP_FlashCacheDisable (void) |
| __STATIC_INLINE void | R_BSP_FlashCacheEnable (void) |
| __STATIC_INLINE void | R_BSP_SecondaryCoreStart (void) |
| void | R_BSP_SoftwareDelay (uint32_t delay, bsp_delay_units_t units) |
| fsp_err_t | R_BSP_GroupIrqWrite (bsp_grp_irq_t irq, void(*p_callback)(bsp_grp_irq_t irq)) |
| void | NMI_Handler (void) |
| fsp_err_t | R_BSP_IpcSemaphoreTake (bsp_ipc_semaphore_handle_t const *const p_semaphore_handle) |
| fsp_err_t | R_BSP_IpcSemaphoreGive (bsp_ipc_semaphore_handle_t const *const p_semaphore_handle) |
| fsp_err_t | R_BSP_IpcNmiEnable (bsp_ipc_nmi_cb_t p_callback) |
| fsp_err_t | R_BSP_IpcNmiRequestSet (void) |
| void | R_BSP_IrqClearPending (IRQn_Type irq) |
| void | R_BSP_IrqCfg (IRQn_Type const irq, uint32_t priority, void *p_context) |
| void | R_BSP_IrqEnableNoClear (IRQn_Type const irq) |
| void | R_BSP_IrqEnable (IRQn_Type const irq) |
| void | R_BSP_IrqDisable (IRQn_Type const irq) |
| void | R_BSP_IrqCfgEnable (IRQn_Type const irq, uint32_t priority, void *p_context) |
| __STATIC_INLINE fsp_err_t | R_BSP_MemoryMirrorStatusGet (mmf_status_t *p_mmf_status) |
| __STATIC_INLINE fsp_err_t | R_BSP_MemoryMirrorAddrSet (const uint32_t addr) |
| void | R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect) |
| void | R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect) |
| void | Reset_Handler (void) |

| | |
|---|---|
| void | Default_Handler (void) |
| void | SystemInit (void) |
| void | R_BSP_WarmStart (bsp_warm_start_event_t event) |

### Detailed Description

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

- BSP Features
- BSP Clock Configuration
- System Interrupts
- Group Interrupts
- External and Peripheral Interrupts
- Error Logging
- BSP Weak Symbols
- Warm Start Callbacks
- Sub Clock Stabilization Wait Callback
- SDRAM Initialization
- C Runtime Initialization
- Register Protection
- Option-Setting Memory
- ID Codes
- TrustZone Security Attribution Registers
- Software Delay
- Trignometric Function
- Digital Signal Processing with 32-bit Multiply-Accumulator
- Octal-SPI Clock Update
- Devices with OFS1.INITECCEN
- Limited D-Cache Support
- Non-Cacheable Buffer Placement Example
- Inter-Processor Communication (IPC) Support
- Configuration

# Overview

### BSP Features

### BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in bsp_clock_cfg.h. These settings are derived from clock configuration information provided from the RA Configuration editor **Clocks** tab.

- Clock configuration is performed prior to initializing the C runtime environment to speed up the startup process, as it is possible to start up on a relatively slow (that is, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.
- The BSP will configure the CMSIS SystemCoreClock variable after clock initialization with the current system clock frequency.

### System Interrupts

As RA MCUs are based on the Arm Cortex-M architecture, the NVIC Nested Vectored Interrupt Controller (NVIC) handles exceptions and interrupt configuration, prioritization and interrupt masking. In the Arm architecture, the NVIC handles exceptions. Some exceptions are known as System Exceptions. System exceptions are statically located at the "top" of the vector table and occupy vector numbers 1 to 15. Vector zero is reserved for the MSP Main Stack Pointer (MSP). The remaining 15 system exceptions are shown below:

- Reset
- NMI
- Cortex-M4 Hard Fault Handler
- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCall Handler
- Cortex-M4 Debug Monitor Handler
- Reserved
- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI and Hard Fault exceptions are enabled out of reset and have fixed priorities. Other exceptions have configurable priorities and some can be disabled.

### Group Interrupts

Group interrupt is the term used to describe the 12 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the RA peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDT Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- VBATT monitor Interrupt
- Oscillation Stop is detected
- NMI pin
- RAM Parity Error
- RAM ECC Error
- MPU Bus Slave Error
- MPU Bus Master Error
- MPU Stack Error
- TrustZone Filter Error A user may enable notification for one or more group interrupts by registering a callback using the BSP API function R_BSP_GroupIrqWrite(). When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

### External and Peripheral Interrupts

User configurable interrupts begin with slot 16. These may be external, or peripheral generated interrupts.

Although the number of available slots for the NVIC interrupt vector table may seem small, the BSP defines up to 512 events that are capable of generating an interrupt. By using Event Mapping, the BSP maps user-enabled events to NVIC interrupts. For an RA6M3 MCU, only 96 of these events may be active at any one time, but the user has flexibility by choosing which events generate the active event.

By allowing the user to select only the events they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and event specific.

For example, on other microcontrollers a standard NVIC interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the RA implementation there is a vector entry for each of the SCI0 events that we are interested in.

### BSP Weak Symbols

You might wonder how the BSP is able to place ISR addresses in the NVIC table without the user having explicitly defined one. All that is required by the BSP is that the interrupt event be given a priority.

This is accomplished through the use of the 'weak' attribute. The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. A weak symbol is one that can be overridden by an accompanying strong reference with the same name. When the BSP declares a function as weak, user code can define the same function and it will be used in place of the BSP function. By defining all possible interrupt sources as weak, the vector table can be built at compile time and any user declarations (strong references) will be used at runtime.

Weak symbols are supported for ELF targets and also for a.out targets when using the GNU assembler and linker.

Note that in CMSIS system.c, there is also a weak definition (and a function body) for the Warm Start callback function R_BSP_WarmStart(). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by the user by copying the body into their user application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

### Warm Start Callbacks

As the BSP is in the process of bringing up the board out of reset, there are three points where the user can request a callback. These are defined as the 'Pre Clock Init', 'Post Clock Init' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as R_BSP_WarmStart(), so it is a simple matter of redefining the function or copying the existing body from CMSIS system.c into the application code to get a callback. R_BSP_WarmStart() takes an event parameter of type bsp_warm_start_event_t which describes the type of warm start callback being made.

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system.c. To use this function just copy this function into your own code and

modify it to meet your needs.

### Sub Clock Stabilization Wait Callback

When Sub-Clock oscillator is populated in the application, the BSP startup code waits for some time(sub-clock stabilization time) to allow Sub-clock to stabilize. Enabling the watchdog (IWDT or WDT) timer with Auto start mode in an application using Sub-clock may cause system to generate Reset or NMI interrupt before reaching the application code if watchdog refresh register is not updated in the configured refresh Window. To overcome this problem a weakly defined callback R_BSP_SubClockStabilizeWait() can be overridden. Redefine the callback function in the application code and add code to update the watchdog refresh register. R_BSP_SubClockStabilizeWait() takes a parameter delay of type uint32_t which describes the time in milliseconds required to stabilize the sub-clock.

### Sub Clock Stabilization Wait After Reset callback

After Power-On-Reset, the BSP startup code may have to wait for some time(sub-clock stabilization time) to allow Sub-clock to stabilize. This can cause problem to RTC in case device is to be reset frequently. If Sub-Clock registers are not initialized during a reset, BSP actually does not have to wait for Sub-clock to stabilize. To overcome this problem, a weakly defined callback R_BSP_SubClockStabilizeWaitAfterReset() is provided. Reimplement the callback function in the application code to determine whether BSP has to wait for stabilization time based on the current reset type. R_BSP_SubClockStabilizeWaitAfterReset() takes a parameter delay of type uint32_t which describes the time in milliseconds required to stabilize the sub-clock.

### SDRAM Initialization

The BSP provides support for usage of external SDRAM modules on MCUs with SDRAM support. SDRAM is enabled and configured in the BSP tab of the RA configuration editor. The default location for initialization is in the 'Post C' warm start callback. If required, the call to R_BSP_SdramInit() can be moved anywhere after clock and pin initialization, but it must only be called once after reset. The BSP will not initialize any memory sections in the SDRAM. The user is responsible for initializing any code or data stored in SDRAM.

Before entering Software Standby or Deep Software Standby, the user must call R_BSP_SdramSelfRefreshEnable() to change from Auto-Refresh to Self-Refresh in order to preserve data during the low power state. No SDRAM access is allowed after this function is called. The user must not place FSP code or data or their wakeup interrupt handling functions into SDRAM to ensure there are no issues around transitions into and out of Software Standby since SDRAM access will be disabled then and trigger a fault if access is requested.

When resuming from Software Standby, the user must call R_BSP_SdramSelfRefreshDisable() to change from Self-Refresh to Auto-Refresh and restore SDRAM access.

When resuming operation after Deep Software Standby or another situation where there is already data present in the SDRAM modules that must be preserved, the user must call R_BSP_SdramInit(false) before pin initialization and then call R_BSP_SdramSelfRefreshDisable() after pins have been configured in order to resume operations with the SDRAM.

### OSPI_B Initialization

The BSP provides support for usage of external or SiP OSPI memories on MCUs with OSPI_B support. OSPI_B intialization is enabled in the BSP tab of the RA configuration editor, but an OSPI_B stack component must also be added. It requires the "Early BSP Initialization" option to be enabled. The default location for initialization is during the BSP_WARM_START_POST_CLOCK event of

R_BSP_WarmStart callback. When enabled, the function specified in the macro BSP_CFG_OSPI_B_STARTUP_FN will be executed by the BSP and any internal memory regions sourced from OSPI_B memory will be initialized. The BSP will not initialize any memory sections in the OSPI memory; the user is responsible for initializing any code or data stored in the OSPI memory.

OSPI must be initialized before other external memories (for example, HyperRAM) if they source initialization data from OSPI. If required, the call to R_BSP_OspiBInit() can be moved anywhere after the clock initialization, but must be called once after reset. Moving initilization to be after the BSP_WARM_START_POST_C event will prevent external memories from automatically being initialized if sourced from OSPI.

On low voltage devices, for example, RA8P1 SiP, the BSP must be configured to the appropriate I/O port driver voltage or instabilities can occur.

### C Runtime Initialization

This BSP configuration allows the user to skip the FSP C runtime initialization code by setting the "C Runtime Initialization" to "Disabled" on the BSP tab of the RA Configuration editor. Disabling this option is useful in cases where a non-standard linker script is being used or other modifications to the runtime initialization are desired. If this macro is disabled, the user must use the 'Post Clock Init' event from the warm start (described above) to run their own runtime initialization code.

### Heap Allocation

The relatively low amount of on-chip SRAM available and lack of memory protection in an MCU means that heap use must be very carefully controlled to avoid memory leaks, overruns and attempted overallocation. Further, many RTOSes provide their own dynamic memory allocation system. For these reasons the default heap size is set at 0 bytes, effectively disabling dynamic memory. If it is required for an application setting a positive value to the "Heap size (bytes)" option in the RA Common configurations on the **BSP** tab will allocate a heap.

*Note*

*When using printf/sprintf (and other variants) to output floating point numbers a heap is required. A minimum size of 0x1000 (4096) bytes is recommended when starting development in this case.*

### Error Logging

When error logging is enabled, the error logging function can be redefined on the command line by defining FSP_ERROR_LOG(err) to the desired function call. The default function implementation is FSP_ERROR_LOG(err)=fsp_error_log(err, **FILE**, **LINE**). This implementation uses the predefined macros **FILE** and **LINE** to help identify the location where the error occurred. Removing the line from the function call can reduce code size when error logging is enabled. Some compilers may support other predefined macros like **FUNCTION**, which could be helpful for customizing the error logger.

### Register Protection

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function doesn't have its register protection settings inadvertently modified.

Each time R_BSP_RegisterProtectDisable() is called, the respective reference counter is incremented.

Each time R_BSP_RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

```
/* Enable writing to protected CGC registers */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* Insert code to modify protected CGC registers. */
/* Disable writing to protected CGC registers */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

### Option-Setting Memory

Option-setting memory includes OFS registers (e.g. OFS1), the debugger ID code (OSIS), block protections settings (BPS/PBPS), and other device-dependent non-volatile options. Option-setting memory is MCU specific, and not all MCUs implement all option-setting registers. Option-setting configurations available on the selected device are configurable in the BSP properties. These configurations are placed in sections to be loaded at the required flash address by the linker.

OFS registers can be individually enabled or disabled. When an OFS register is disabled, its flash address will not be erased or programmed. The existing settings on the device will not be changed. For safety purpose, OFS registers that could limit or disable debug or ROM write access (like OSIS, BANKSEL, BPS, PBPS etc.) are disabled by default. Please note:

- Secure OFS registers can only be set by Secure/Flat projects. Secure OFS register settings are ignored in Non-Secure projects.
- For multiple-core use case, OFS registers will only be set by CPU0 projects.
- For MCUBoot use case, OFS registers will only be set by bootloader projects.

### ID Codes

The ID code is a 16-byte value that can be used to protect the MCU from being connected to a debugger or from connecting in Serial Boot Mode. There are different settings that can be set for the ID code; please refer to the hardware manual for your device for available options.

On MCUs that support TrustZone, option-setting registers are placed in a different locations for Non-Secure projects than for Secure or Flat projects. This is handled automatically by the BSP and linker scripts.

All *_SEL registers default to allowing both Secure and Non-Secure access unless otherwise noted here. If block protection is configured in a Secure project, the BSP sets the corresponding configuration to Secure access only by updating the corresponding *_SEL register. Similarly, the LVD related settings in the OFSn_SEL registers are automatically set to Secure if the corresponding LVD monitor is used in the Secure project.

### TrustZone Security Attribution Registers

On MCUs that support TrustZone, Security Attribution Registers for modules used in the Secure project are configured to allow Secure access only as part of the startup code of the Secure project. This logic is skipped for Flat projects.

### Software Delay

Implements a blocking software delay. A delay can be specified in microseconds, milliseconds or seconds. The delay is implemented based on the system clock rate.

```
 /* Delay at least 1 second. Depending on the number of wait states required for the
region of memory
  * that the software_delay_loop has been linked in this could take longer. The
default is 4 cycles per loop.
  * This can be modified by redefining DELAY_LOOP_CYCLES. BSP_DELAY_UNITS_SECONDS,
BSP_DELAY_UNITS_MILLISECONDS,
  * and BSP_DELAY_UNITS_MICROSECONDS can all be used with R_BSP_SoftwareDelay. */
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

### Trignometric Function

Implements Trignometric math inline functions utilizing TFU hardware. These functions can calculate sine, cosine, arctangent and hypotenuse. The trigonometric library functions sinf(), cosf(), atan2f(), and hypotf() can be mapped to respective TFU functions by enabling TFU Mathlib property in FSP Configuration tool. Extended functions sincosf() and atan2hypotf() are also available when the TFU Mathlib property is enabled in the RA Configuration editor.

TFU functions are not reentrant. Disable the TFU Mathlib property in RA Configuration editor if reentrant access to trigonometric library functions is required.

*Note*

*Refer to the MCU hardware user's manual or datasheet to determine if it has TFU support.*

### Digital Signal Processing With 32-bit Multiply-Accumulator

Implements DSP (digital signal processing) functions via MACL (32-bit Multiply-Accumulator) hardware. These functions will support CMSIS DSP APIs to perform the calculation, the activation of MACL can be controlled by stack MACL (rm_cmsis_dsp) in the RA Configuration editor.

When stack MACL (rm_cmsis_dsp) is added, the CMSIS DSP APIs will generate normal functions which overrides weak functions of Arm and use hardware for calculating. When stack MACL (rm_cmsis_dsp) is not added, CMSIS DSP APIs will perform the calculation by using software.

For examples, how to use the CMSIS DSP APIs refer to the link: https://github.com/ARM-software/CMSIS-DSP/tree/main/Examples/ARM.

CMSIS APIs supported by MACL list:

| No | CMSIS DSP API | MACL BSP API | Description |
|---|---|---|---|
| 1 | arm_mult_q31 | R_BSP_MaclMulQ31 | Q31 vector multiplication |
| 2 | arm_scale_q31 | R_BSP_MaclScaleQ31 | Multiplies a Q31 vector by a scalar |
| 3 | arm_mat_mult_q31 | R_BSP_MaclMatMultQ31 | Q31 matrix multiplication |
| 4 | arm_mat_vec_mult_q31 | R_BSP_MaclMatVecMul | Q31 matrix and vector |

| | | | |
|---|---|---|---|
| | | Q31 | multiplication |
| 5 | arm_mat_scale_q31 | R_BSP_MaclMatScaleQ31 | Q31 matrix scaling |
| 6 | arm_biquad_cascade_df1_q31 | R_BSP_MaclBiquadCsdDf1Q31 | Processing function for the Q31 Biquad cascade filter |
| 7 | arm_conv_partial_q31 | R_BSP_MaclConvPartialQ31 | Partial convolution of Q31 sequences |
| 8 | arm_conv_q31 | R_BSP_MaclConvQ31 | Convolution of Q31 sequences |
| 9 | arm_correlate_q31 | R_BSP_MaclCorrelateQ31 | Correlation of Q31 sequences |
| 10 | arm_fir_decimate_q31 | R_BSP_MaclFirDecimateQ31 | Processing function for the Q31 FIR decimator |
| 11 | arm_fir_interpolate_q31 | R_BSP_MaclFirInterpolateQ31 | Processing function for the Q31 FIR interpolator |
| 12 | arm_fir_q31 | R_BSP_MaclFirQ31 | Processing function for Q31 FIR filter |
| 13 | arm_fir_sparse_q31 | R_BSP_MaclFirSparseQ31 | Processing function for the Q31 sparse FIR filter |
| 14 | arm_lms_norm_q31 | R_BSP_MaclLmsNormQ31 | Processing function for Q31 normalized LMS filter |
| 15 | arm_lms_q31 | R_BSP_MaclLmsQ31 | Processing function for Q31 LMS filter |

*Note*

*Refer to the MCU hardware user's manual or datasheet to determine if it has MACL support.*

### Critical Section Macros

Implements a critical section. Some MCUs (MCUs with the BASEPRI register) support allowing high priority interrupts to execute during critical sections. On these MCUs, interrupts with priority less than or equal to BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION are not serviced in critical sections. Interrupts with higher priority than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION still execute in critical sections.

```
   FSP_CRITICAL_SECTION_DEFINE;
/* Store the current interrupt posture. */
FSP_CRITICAL_SECTION_ENTER;
/* Interrupts cannot run in this section unless their priority is less than
BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. */
```

```
/* Restore saved interrupt posture. */
FSP_CRITICAL_SECTION_EXIT;
```

### OctaClock Update

Supports changing the Octal-SPI Clock (OCTACLK) during runtime if supported by the MCU. The OCTACLK source and clock divisor can be updated. It is user's responsibility to ensure the selected clock source is running before attempting to update OCTACLK.

### Devices with OFS1.INITECCEN

By default, FSP disables ECC for the caches and TCM with OFS1.INITECCEN. For best performance, it is recommended to keep ECC for cache and TCM disabled.

*Note*

*When ECC is enabled for TCM, a full and unconditional zero-initialization of all TCM memory will occur at startup of the first project after reset. This may be either a bootloader if present, or a Flat or Secure application. Because of this, any noinit section contained within TCM will be functionally a zero-initialized section, if ECC is enabled. This is an unavoidable requirement of ECC initialization. Also see Cache ECC with FSP.*

### Devices with LVOCR

By default, FSP disables I/O port low voltage mode for devices with the LVOCR register. Presently, RA8 devices are the only RA devices with the LVOCR register, which configures I/O ports based on VCC and VCC2 voltage levels. For cases where VCC and/or VCC2 will be less than 2.7 V for these devices, the I/O port low voltage mode should be enabled for VCC and/or VCC2.

*Note*

*If the I/O port low voltage mode is not enabled for VCC and/or VCC2 when less than 2.7 V, device electrical characteristics cannot be guaranteed. Do not enable I/O port low voltage mode for VCC or VCC2 when they are greater than or equal to 2.7 V.*

### Sealing the Main Stack (TrustZone Secure Projects)

In TrustZone secure projects, the BSP seals the main stack by placing the value 0xFEF5EDA5 above the stack top. For more information, refer to section 3.5 "Sealing a Stack" in "Secure software guidelines for ARMv8-M": https://developer.arm.com/documentation/100720/0300.

## Limited D-Cache Support

For MCUs with Cortex-M85 cores with D-Cache, limited support is available for enabling the D-Cache and automatically configuring predefined non-cacheable regions via the MPU during BSP initialization. For these MCUs, D-Cache is disabled by default because certain existing drivers do not support data coherency with D-Cache enabled. Enabling the D-Cache requires that data coherency be considered in any circumstance where a core interacts with other bus members.

### Non-Cacheable Buffer Placement Example

Warning

**The predefined non-cacheable sections are only understood as non-cacheable**

**within their respective security state when TrustZone is used. This is due to the MPU configuration done by FSP and how Armv8-M chooses the MPU which matches the current security state for determining the memory attributes of a data access. Cache incoherency will occur if references to these sections are passed between security states. For example, passing a reference from the Non-secure application non-cacheable section to the Secure application will result in the Secure application treating the location as cacheable.**
**The predefined non-cacheable sections are only understood as non-cacheable within their respective core. In dual-core devices, each core has a separate MPU which may configure its own regions and attributes, so these sections cannot be used for coherent interprocessor communications in the case that D-Cache is enabled. Use the IPC peripheral instead.**

The predefined non-cacheable regions configured by the MPU when D-Cache is enabled can be used to contain data that should not be cached, ensuring data coherency for that data. To use the predefined non-cacheable regions, place the data into the corresponding non-cacheable section defined by the linker script for the chosen toolchain. The predefined non-cacheable regions are not initialized by the BSP.

Below are two examples of placing buffers in non-cacheable regions:

```
uint8_t uncached_uninitialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_noinit_no
cache");

uint8_t uncached_zero_initialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_nocach
e");
```

*Note*

*Prior to FSP 6.0.0, the .ram_nocache and .sdram_nocache sections were treated as uninitialized sections. In v6.0.0 and up they default to zero initialization during startup. Use the new section names containing _noinit_ for placement into an uninitialized non-cacheable section.*

Section names differ by data region and compiler. Names predefined by the BSP are shown below.

| Region | Initialization | Name (GCC, IAR, LLVM) | Name (AC6) |
|---|---|---|---|
| SRAM | Uninitialized | .ram_noinit_nocache | .bss.ram_noinit_nocache |
| SRAM | Zero | .ram_nocache | .bss.ram_nocache |
| SDRAM | Uninitialized | .sdram_noinit_nocache | .bss.sdram_noinit_nocache |
| SDRAM | Zero | .sdram_nocache | .bss.sdram_nocache |
| OSPI0 CS0 | Uninitialized | .ospi0_cs0_noinit_nocache | .bss.ospi0_cs0_noinit_nocache |
| OSPI0 CS0 | Zero | .ospi0_cs0_nocache | .bss.ospi0_cs0_nocache |
| OSPI1 CS0 | Uninitialized | .ospi1_cs0_noinit_nocache | .bss.ospi1_cs0_noinit_nocache |
| OSPI1 CS0 | Zero | .ospi1_cs0_nocache | .bss.ospi1_cs0_nocache |

# Inter-Processor Communication (IPC) Support

*Note*

*For using IPC message FIFOs and maskable interrupts see the HAL IPC driver: Overview*

For dual-core MCUs with an IPC peripheral, the BSP provides APIs to use IPC NMIs and semaphores.

### NMI Support

In order to configure receiving IPC NMIs from the opposing core the user should call R_BSP_IpcNmiEnable to enable the interrupt and setup a callback for interrupt processing. On the other hand R_BSP_IpcNmiRequestSet should be called to trigger an interrupt on the opposing core.

### Semaphore Support

R_BSP_IpcSemaphoreTake and R_BSP_IpcSemaphoreGive should be used to manage any IPC semaphore. The semaphore_handle should be linked to a valid hardware semaphore number.

A IPC semaphore stack is provided which can be used to create semaphore handles.

### Configurations for System > IPC Semaphore

This module can be added to the Stacks tab via New Stack > System > IPC Semaphore.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ipc_semaphore0 | Module name. |
| Semaphore Number | MCU Specific Options | | Select the IPC semaphore number. This semaphore number will determine which hardware semaphore register is used. |

# Configuration

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated when the project is built and/or when the Generate Project Content button is clicked in the RA Configuration editor.

### Build Time Configurations for fsp_common

The following build time configurations are defined in fsp_cfg/bsp/bsp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Main stack size (bytes) | Value must be an integer multiple of 8 and between 8 and 0xFFFFFFFF | 0x400 | Set the size of the main program stack.<br><br>NOTE: This entry is for |

| | | | |
|---|---|---|---|
| | | | the main stack. When using an RTOS, thread stacks can be configured in the properties for each thread. |
| Heap size (bytes) | Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF. | 0 | The main heap is disabled by default. Set the heap size to a positive integer divisible by 8 to enable it.<br><br>A minimum of 4K (0x1000) is recommended if standard library functions are to be used. |
| Bootloader Secondary XIP | • Disabled<br>• Enabled | Disabled | This bootloader application project wants to be linked to Secondary XIP area |
| MCU Vcc (mV) | Value must between 0 and 5500 (5.5V) | 3300 | Some peripherals require different settings based on the supplied voltage. Entering Vcc here (in mV) allows the relevant driver modules to configure the associated peripherals accordingly. |
| Parameter checking | • Enabled<br>• Disabled | Disabled | When enabled, parameter checking for the BSP is turned on. In addition, any modules whose parameter checking configuration is set to 'Default (BSP)' will perform parameter checking as well. |
| Assert Failures | • Return FSP_ERR_ASSERTION<br>• Call fsp_error_log then Return FSP_ERR_ASSERTION<br>• Use assert() to Halt Execution<br>• Disable checks | Return FSP_ERR_ASSERTION | Define the behavior of the FSP_ASSERT() macro. |

| | | | |
|---|---|---|---|
| | that would return FSP_ERR_ASSERTION | | |
| Clock Registers not Reset Values during Startup | • Disabled<br>• Enabled | Disabled | If enabled, registers are assumed to be set to their reset value during startup. Enable this if another application such as a bootloader or Secure project has already configured the clocks before the startup code runs. |
| Main Oscillator Populated | • Populated<br>• Not Populated | Populated | Select whether or not there is a main oscillator (XTAL) on the board. This setting can be overridden in board_cfg.h. |
| PFS Protect | • Disabled<br>• Enabled | Enabled | Keep the PFS registers locked when they are not being modified. If disabled they will be unlocked during startup. |
| C Runtime Initialization | • Enabled<br>• Disabled | Enabled | Select if the C runtime initialization in the BSP is to be used. If disabled, use the BSP_WARM_START_POST_CLOCK event to run user defined equivalent. |
| Early BSP Initialization | • Enabled<br>• Disabled | Disabled | Enable this option to use BSP functions before C runtime initialization (BSP_WARM_START_RESET or BSP_WARM_START_POST_CLOCK). |
| Main Oscillator Clock Source | • External Oscillator<br>• Crystal or Resonator | Crystal or Resonator | Select the main oscillator clock source. This setting can be overridden in board_cfg.h |
| Subclock Populated | • Populated<br>• Not Populated | Populated | Select whether or not there is a subclock crystal on the board. This setting can be overridden in board_cfg.h. |

| | | | |
|---|---|---|---|
| Subclock Drive (Drive capacitance availability varies by MCU) | • Standard/Normal mode<br>• Low/Low power mode 1<br>• Low power mode 2<br>• Low power mode 3 | Standard/Normal mode | Select the subclock oscillator drive capacitance. This setting can be overridden in board_cfg.h |
| Subclock Stabilization Time (ms) | Value must between 0 and 10000 | 1000 | Select the subclock oscillator stabilization time. This is only used in the startup code if the subclock is selected as the system clock on the Clocks tab or if the HOCO FLL function is enabled. This setting can be overridden in board_cfg.h |

**Modules**

RA0E1

RA0E2

RA0L1

RA2A1

RA2A2

RA2E1

RA2E2

RA2E3

RA2L1

RA2L2

RA2T1

RA4C1

RA4E1

RA4E2

| |
|---|
| RA4L1 |
| RA4M1 |
| RA4M2 |
| RA4M3 |
| RA4T1 |
| RA4W1 |
| RA6E1 |
| RA6E2 |
| RA6M1 |
| RA6M2 |
| RA6M3 |
| RA6M4 |
| RA6M5 |
| RA6T1 |
| RA6T2 |
| RA6T3 |
| RA8D1 |
| RA8D2 |
| RA8E1 |
| RA8E2 |
| RA8M1 |
| RA8M2 |
| RA8P1 |
| RA8T1 |
| |

| | |
|---|---|
| | RA8T2 |

## Data Structures

| | |
|---|---|
| struct | bsp_ipc_semaphore_handle_t |
| struct | mmf_status_t |

## Macros

| | |
|---|---|
| #define | BSP_IRQ_DISABLED |
| #define | FSP_LOG_PRINT(X) |
| #define | FSP_RETURN(err) |
| #define | FSP_ERROR_LOG(err) |
| #define | FSP_ASSERT(a) |
| #define | FSP_ERROR_RETURN(a, err) |
| #define | FSP_CRITICAL_SECTION_ENTER |
| #define | FSP_CRITICAL_SECTION_EXIT |
| #define | FSP_INVALID_VECTOR |
| #define | BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(x) |
| #define | BSP_STACK_ALIGNMENT |
| #define | MEMORY_MIRROR_REG_KEY |
| #define | R_BSP_MODULE_START(ip, channel) |
| #define | R_BSP_MODULE_STOP(ip, channel) |

## Typedefs

| | |
|---|---|
| typedef void(* | bsp_ipc_nmi_cb_t) (void) |

## Enumerations

| | |
|---|---|
| enum | bsp_warm_start_event_t |
| enum | fsp_priv_clock_t |
| enum | fsp_priv_source_clock_t |
| enum | bsp_delay_units_t |

| | |
|---|---|
| enum | bsp_grp_irq_t |
| enum | mmf_state_t |
| enum | bsp_reg_protect_t |
| enum | fsp_ip_t |
| enum | fsp_signal_t |

### Variables

| | |
|---|---|
| uint32_t SystemCoreClock | BSP_SECTION_EARLY_INIT |

## Data Structure Documentation

### ◆ bsp_ipc_semaphore_handle_t

| struct bsp_ipc_semaphore_handle_t | | |
|---|---|---|
| Semaphore handle for IPC semaphores. | | |
| Data Fields | | |
| uint8_t | semaphore_num | Semaphore number, controls which IPCSEMn register is used. |

### ◆ mmf_status_t

| struct mmf_status_t |
|---|
| Status instance of Memory Mirror Function. |

## Macro Definition Documentation

### ◆ BSP_IRQ_DISABLED

| #define BSP_IRQ_DISABLED |
|---|
| Used to signify that an ELC event is not able to be used as an interrupt. |

### ◆ FSP_LOG_PRINT

| #define FSP_LOG_PRINT ( X) |
|---|
| Macro that can be defined in order to enable logging in FSP modules. |

### ◆ FSP_RETURN

| #define FSP_RETURN ( err) |
| --- |
| Macro to log and return error without an assertion. |

### ◆ FSP_ERROR_LOG

| #define FSP_ERROR_LOG ( err) |
| --- |
| This function is called before returning an error code. To stop on a runtime error, define fsp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function. |

### ◆ FSP_ASSERT

| #define FSP_ASSERT ( a) |
| --- |
| Default assertion calls FSP_ERROR_RETURN if condition "a" is false. Used to identify incorrect use of API's in FSP functions. |

### ◆ FSP_ERROR_RETURN

| #define FSP_ERROR_RETURN ( a, err ) |
| --- |
| All FSP error codes are returned using this macro. Calls FSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in FSP functions. |

### ◆ FSP_CRITICAL_SECTION_ENTER

| #define FSP_CRITICAL_SECTION_ENTER |
| --- |
| This macro temporarily saves the current interrupt state and disables interrupts. |

### ◆ FSP_CRITICAL_SECTION_EXIT

| #define FSP_CRITICAL_SECTION_EXIT |
| --- |
| This macro restores the previously saved interrupt state, reenabling interrupts. |

### ◆ FSP_INVALID_VECTOR

| #define FSP_INVALID_VECTOR |
| --- |
| Used to signify that the requested IRQ vector is not defined in this system. |

### ◆ BSP_CFG_HANDLE_UNRECOVERABLE_ERROR

| #define BSP_CFG_HANDLE_UNRECOVERABLE_ERROR ( x) |
| --- |
| In the event of an unrecoverable error the BSP will by default call the __BKPT() intrinsic function which will alert the user of the error. The user can override this default behavior by defining their own BSP_CFG_HANDLE_UNRECOVERABLE_ERROR macro. |

### ◆ BSP_STACK_ALIGNMENT

| #define BSP_STACK_ALIGNMENT |
| --- |
| Stacks (and heap) must be sized and aligned to an integer multiple of this number. |

### ◆ MEMORY_MIRROR_REG_KEY

| #define MEMORY_MIRROR_REG_KEY |
| --- |
| Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. |

### ◆ R_BSP_MODULE_START

| #define R_BSP_MODULE_START ( ip, channel ) |
| --- |
| Cancels the module stop state.<br><br>**Parameters**<br><br>ip — fsp_ip_t enum value for the module to be stopped<br><br>channel — The channel. Use channel 0 for modules without channels. |

### ◆ R_BSP_MODULE_STOP

| #define R_BSP_MODULE_STOP ( ip, channel ) |
| --- |
| Enables the module stop state.<br><br>**Parameters**<br><br>ip — fsp_ip_t enum value for the module to be stopped<br><br>channel — The channel. Use channel 0 for modules without channels. |

## Typedef Documentation

### ◆ bsp_ipc_nmi_cb_t

| typedef void(* bsp_ipc_nmi_cb_t) (void) |
|---|
| IPC NMI callback type. |

## Enumeration Type Documentation

### ◆ bsp_warm_start_event_t

| enum bsp_warm_start_event_t | |
|---|---|
| Different warm start entry locations in the BSP. | |
| Enumerator | |
| BSP_WARM_START_RESET | Called almost immediately after reset. No C runtime environment, clocks, or IRQs. |
| BSP_WARM_START_POST_CLOCK | Called after clock initialization. No C runtime environment or IRQs. |
| BSP_WARM_START_POST_C | Called after clocks and C runtime environment have been set up. |

### ◆ fsp_priv_clock_t

| enum fsp_priv_clock_t | |
|---|---|
| Enumerator | |
| FSP_PRIV_CLOCK_UNUSED | Sentinel value for unused clock. |

### ◆ fsp_priv_source_clock_t

| enum fsp_priv_source_clock_t | |
|---|---|
| Enumerator | |
| FSP_PRIV_CLOCK_HOCO | The high speed on chip oscillator. |
| FSP_PRIV_CLOCK_MOCO | The middle speed on chip oscillator. |
| FSP_PRIV_CLOCK_LOCO | The low speed on chip oscillator. |
| FSP_PRIV_CLOCK_MAIN_OSC | The main oscillator. |
| FSP_PRIV_CLOCK_SUBCLOCK | The subclock oscillator. |
| FSP_PRIV_CLOCK_PLL | The PLL output. |
| FSP_PRIV_CLOCK_PLL1P | The PLL1P output. |
| FSP_PRIV_CLOCK_PLL2 | The PLL2 output. |
| FSP_PRIV_CLOCK_PLL2P | The PLL2P output. |
| FSP_PRIV_CLOCK_PLL1Q | The PLL1Q output. |
| FSP_PRIV_CLOCK_PLL1R | The PLL1R output. |
| FSP_PRIV_CLOCK_PLL2Q | The PLL2Q output. |
| FSP_PRIV_CLOCK_PLL2R | The PLL2R output. |

### ◆ bsp_delay_units_t

| enum bsp_delay_units_t | |
|---|---|
| Available delay units for R_BSP_SoftwareDelay(). These are ultimately used to calculate a total # of microseconds | |
| Enumerator | |
| BSP_DELAY_UNITS_SECONDS | Requested delay amount is in seconds. |
| BSP_DELAY_UNITS_MILLISECONDS | Requested delay amount is in milliseconds. |
| BSP_DELAY_UNITS_MICROSECONDS | Requested delay amount is in microseconds. |

### ◆ bsp_grp_irq_t

| enum bsp_grp_irq_t | |
|---|---|
| Which interrupts can have callbacks registered. | |
| Enumerator | |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_VBATT | VBATT monitor interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_RAM_PARITY | RAM Parity Error. |
| BSP_GRP_IRQ_RAM_ECC | RAM ECC Error. |
| BSP_GRP_IRQ_MPU_BUS_SLAVE | MPU Bus Slave Error. |
| BSP_GRP_IRQ_MPU_BUS_MASTER | MPU Bus Master Error. |
| BSP_GRP_IRQ_MPU_STACK | MPU Stack Error. |
| BSP_GRP_IRQ_TRUSTZONE | MPU Stack Error. |
| BSP_GRP_IRQ_CACHE_PARITY | MPU Stack Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |

| | |
|---|---|
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_SOSC_STOP_DETECT | Sub Oscillation stop is detected. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCAL_MEMORY | Local Memory Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_FPU | FPU Exception. |
| BSP_GRP_IRQ_MRC | MRAM Code read error. |
| BSP_GRP_IRQ_MRE | MRAM Extra read error. |
| BSP_GRP_IRQ_IPC | IPC Interrupt. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |

| | |
|---|---|
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |

| | |
|---|---|
| BSP_GRP_IRQ_SOSC_STOP_DETECT | Sub Oscillation stop is detected. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCAL_MEMORY | Local Memory Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_FPU | FPU Exception. |
| BSP_GRP_IRQ_MRC | MRAM Code read error. |
| BSP_GRP_IRQ_MRE | MRAM Extra read error. |
| BSP_GRP_IRQ_IPC | IPC Interrupt. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_SOSC_STOP_DETECT | Sub Oscillation stop is detected. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCAL_MEMORY | Local Memory Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_FPU | FPU Exception. |
| BSP_GRP_IRQ_MRC | MRAM Code read error. |

| | |
|---|---|
| BSP_GRP_IRQ_MRE | MRAM Extra read error. |
| BSP_GRP_IRQ_IPC | IPC Interrupt. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_IWDT_ERROR | IWDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_WDT_ERROR | WDT underflow/refresh error has occurred. |
| BSP_GRP_IRQ_LVD1 | Voltage monitoring 1 interrupt. |
| BSP_GRP_IRQ_LVD2 | Voltage monitoring 2 interrupt. |
| BSP_GRP_IRQ_SOSC_STOP_DETECT | Sub Oscillation stop is detected. |
| BSP_GRP_IRQ_OSC_STOP_DETECT | Oscillation stop is detected. |
| BSP_GRP_IRQ_NMI_PIN | NMI Pin interrupt. |
| BSP_GRP_IRQ_MPU_BUS_TZF | MPU Bus or TrustZone Filter Error. |
| BSP_GRP_IRQ_COMMON_MEMORY | SRAM ECC or SRAM Parity Error. |
| BSP_GRP_IRQ_LOCAL_MEMORY | Local Memory Error. |
| BSP_GRP_IRQ_LOCKUP | LockUp Error. |
| BSP_GRP_IRQ_FPU | FPU Exception. |
| BSP_GRP_IRQ_MRC | MRAM Code read error. |

| | |
|---|---|
| BSP_GRP_IRQ_MRE | MRAM Extra read error. |
| BSP_GRP_IRQ_IPC | IPC Interrupt. |

### ◆ mmf_state_t

| enum mmf_state_t |
|---|
| Enum for state of Memory Mirror Function. |

### ◆ bsp_reg_protect_t

| enum bsp_reg_protect_t | |
|---|---|
| The different types of registers that can be protected. | |
| Enumerator | |
| BSP_REG_PROTECT_CGC | Enables writing to the registers related to the clock generation circuit. |
| BSP_REG_PROTECT_OM_LPC_BATT | Enables writing to the registers related to operating modes, low power consumption, and battery backup function. |
| BSP_REG_PROTECT_LVD | Enables writing to the registers related to the LVD: LVCMPCR, LVDLVLR, LVD1CR0, LVD1CR1, LVD1SR, LVD2CR0, LVD2CR1, LVD2SR. |
| BSP_REG_PROTECT_SAR | Enables writing to the registers related to the security function. |
| BSP_REG_PROTECT_RESET | Enables writing to the registers related to reset control. |

### ◆ fsp_ip_t

| enum fsp_ip_t | |
|---|---|
| Available modules. | |
| Enumerator | |
| FSP_IP_CFLASH | Code Flash. |
| FSP_IP_DFLASH | Data Flash. |
| FSP_IP_RAM | RAM. |

| | |
|---|---|
| FSP_IP_LVD | Low Voltage Detection. |
| FSP_IP_CGC | Clock Generation Circuit. |
| FSP_IP_LPM | Low Power Modes. |
| FSP_IP_FCU | Flash Control Unit. |
| FSP_IP_ICU | Interrupt Control Unit. |
| FSP_IP_DMAC | DMA Controller. |
| FSP_IP_DTC | Data Transfer Controller. |
| FSP_IP_IOPORT | I/O Ports. |
| FSP_IP_PFS | Pin Function Select. |
| FSP_IP_ELC | Event Link Controller. |
| FSP_IP_MPU | Memory Protection Unit. |
| FSP_IP_MSTP | Module Stop. |
| FSP_IP_MMF | Memory Mirror Function. |
| FSP_IP_KEY | Key Interrupt Function. |
| FSP_IP_CAC | Clock Frequency Accuracy Measurement Circuit. |
| FSP_IP_DOC | Data Operation Circuit. |
| FSP_IP_CRC | Cyclic Redundancy Check Calculator. |
| FSP_IP_SCI | Serial Communications Interface. |
| FSP_IP_IIC | I2C Bus Interface. |
| FSP_IP_SPI | Serial Peripheral Interface. |
| FSP_IP_CTSU | Capacitive Touch Sensing Unit. |
| FSP_IP_SCE | Secure Cryptographic Engine. |
| FSP_IP_SLCDC | Segment LCD Controller. |
| FSP_IP_AES | Advanced Encryption Standard. |

| | |
|---|---|
| FSP_IP_TRNG | True Random Number Generator. |
| FSP_IP_FCACHE | Flash Cache. |
| FSP_IP_SRAM | SRAM. |
| FSP_IP_ADC | A/D Converter. |
| FSP_IP_DAC | 12-Bit D/A Converter |
| FSP_IP_TSN | Temperature Sensor. |
| FSP_IP_DAAD | D/A A/D Synchronous Unit. |
| FSP_IP_ACMPHS | High Speed Analog Comparator. |
| FSP_IP_ACMPLP | Low Power Analog Comparator. |
| FSP_IP_OPAMP | Operational Amplifier. |
| FSP_IP_SDADC | Sigma Delta A/D Converter. |
| FSP_IP_RTC | Real Time Clock. |
| FSP_IP_WDT | Watch Dog Timer. |
| FSP_IP_IWDT | Independent Watch Dog Timer. |
| FSP_IP_GPT | General PWM Timer. |
| FSP_IP_POEG | Port Output Enable for GPT. |
| FSP_IP_OPS | Output Phase Switch. |
| FSP_IP_AGT | Asynchronous General-Purpose Timer. |
| FSP_IP_CAN | Controller Area Network. |
| FSP_IP_IRDA | Infrared Data Association. |
| FSP_IP_QSPI | Quad Serial Peripheral Interface. |
| FSP_IP_USBFS | USB Full Speed. |
| FSP_IP_SDHI | SD/MMC Host Interface. |
| FSP_IP_SRC | Sampling Rate Converter. |

| | |
|---|---|
| FSP_IP_SSI | Serial Sound Interface. |
| FSP_IP_DALI | Digital Addressable Lighting Interface. |
| FSP_IP_ETHER | Ethernet MAC Controller. |
| FSP_IP_EDMAC | Ethernet DMA Controller. |
| FSP_IP_EPTPC | Ethernet PTP Controller. |
| FSP_IP_PDC | Parallel Data Capture Unit. |
| FSP_IP_GLCDC | Graphics LCD Controller. |
| FSP_IP_DRW | 2D Drawing Engine |
| FSP_IP_JPEG | JPEG. |
| FSP_IP_DAC8 | 8-Bit D/A Converter |
| FSP_IP_USBHS | USB High Speed. |
| FSP_IP_OSPI | Octa Serial Peripheral Interface. |
| FSP_IP_CEC | HDMI CEC. |
| FSP_IP_TFU | Trigonometric Function Unit. |
| FSP_IP_IIRFA | IIR Filter Accelerator. |
| FSP_IP_CANFD | CAN-FD. |
| FSP_IP_ULPT | Ultra Low Power Timer ULPT. |
| FSP_IP_SAU | Serial Array Unit. |
| FSP_IP_IICA | Serial Interface IICA. |
| FSP_IP_UARTA | Serial Interface UARTA. |
| FSP_IP_TAU | Timer Array Unit. |
| FSP_IP_TML | 32-bit Interval Timer |
| FSP_IP_MACL | 32-bit Multiply-Accumulator |
| FSP_IP_USBCC | USB Type-C Controller. |

### ◆ fsp_signal_t

| enum fsp_signal_t | |
|---|---|
| Signals that can be mapped to an interrupt. | |
| Enumerator | |
| FSP_SIGNAL_ADC_COMPARE_MATCH | ADC COMPARE MATCH. |
| FSP_SIGNAL_ADC_COMPARE_MISMATCH | ADC COMPARE MISMATCH. |
| FSP_SIGNAL_ADC_SCAN_END | ADC SCAN END. |
| FSP_SIGNAL_ADC_SCAN_END_B | ADC SCAN END B. |
| FSP_SIGNAL_ADC_WINDOW_A | ADC WINDOW A. |
| FSP_SIGNAL_ADC_WINDOW_B | ADC WINDOW B. |
| FSP_SIGNAL_AES_RDREQ | AES RDREQ. |
| FSP_SIGNAL_AES_WRREQ | AES WRREQ. |
| FSP_SIGNAL_AGT_COMPARE_A | AGT COMPARE A. |
| FSP_SIGNAL_AGT_COMPARE_B | AGT COMPARE B. |
| FSP_SIGNAL_AGT_INT | AGT INT. |
| FSP_SIGNAL_CAC_FREQUENCY_ERROR | CAC FREQUENCY ERROR. |
| FSP_SIGNAL_CAC_MEASUREMENT_END | CAC MEASUREMENT END. |
| FSP_SIGNAL_CAC_OVERFLOW | CAC OVERFLOW. |
| FSP_SIGNAL_CAN_ERROR | CAN ERROR. |
| FSP_SIGNAL_CAN_FIFO_RX | CAN FIFO RX. |
| FSP_SIGNAL_CAN_FIFO_TX | CAN FIFO TX. |
| FSP_SIGNAL_CAN_MAILBOX_RX | CAN MAILBOX RX. |
| FSP_SIGNAL_CAN_MAILBOX_TX | CAN MAILBOX TX. |
| FSP_SIGNAL_CGC_MOSC_STOP | CGC MOSC STOP. |
| FSP_SIGNAL_LPM_SNOOZE_REQUEST | LPM SNOOZE REQUEST. |

| | |
|---|---|
| FSP_SIGNAL_LVD_LVD1 | LVD LVD1. |
| FSP_SIGNAL_LVD_LVD2 | LVD LVD2. |
| FSP_SIGNAL_VBATT_LVD | VBATT LVD. |
| FSP_SIGNAL_LVD_VBATT | LVD VBATT. |
| FSP_SIGNAL_ACMPHS_INT | ACMPHS INT. |
| FSP_SIGNAL_ACMPLP_INT | ACMPLP INT. |
| FSP_SIGNAL_CTSU_END | CTSU END. |
| FSP_SIGNAL_CTSU_READ | CTSU READ. |
| FSP_SIGNAL_CTSU_WRITE | CTSU WRITE. |
| FSP_SIGNAL_DALI_DEI | DALI DEI. |
| FSP_SIGNAL_DALI_CLI | DALI CLI. |
| FSP_SIGNAL_DALI_SDI | DALI SDI. |
| FSP_SIGNAL_DALI_BPI | DALI BPI. |
| FSP_SIGNAL_DALI_FEI | DALI FEI. |
| FSP_SIGNAL_DALI_SDI_OR_BPI | DALI SDI OR BPI. |
| FSP_SIGNAL_DMAC_INT | DMAC INT. |
| FSP_SIGNAL_DOC_INT | DOC INT. |
| FSP_SIGNAL_DRW_INT | DRW INT. |
| FSP_SIGNAL_DTC_COMPLETE | DTC COMPLETE. |
| FSP_SIGNAL_DTC_END | DTC END. |
| FSP_SIGNAL_EDMAC_EINT | EDMAC EINT. |
| FSP_SIGNAL_ELC_SOFTWARE_EVENT_0 | ELC SOFTWARE EVENT 0. |
| FSP_SIGNAL_ELC_SOFTWARE_EVENT_1 | ELC SOFTWARE EVENT 1. |
| FSP_SIGNAL_EPTPC_IPLS | EPTPC IPLS. |

| | |
|---|---|
| FSP_SIGNAL_EPTPC_MINT | EPTPC MINT. |
| FSP_SIGNAL_EPTPC_PINT | EPTPC PINT. |
| FSP_SIGNAL_EPTPC_TIMER0_FALL | EPTPC TIMER0 FALL. |
| FSP_SIGNAL_EPTPC_TIMER0_RISE | EPTPC TIMER0 RISE. |
| FSP_SIGNAL_EPTPC_TIMER1_FALL | EPTPC TIMER1 FALL. |
| FSP_SIGNAL_EPTPC_TIMER1_RISE | EPTPC TIMER1 RISE. |
| FSP_SIGNAL_EPTPC_TIMER2_FALL | EPTPC TIMER2 FALL. |
| FSP_SIGNAL_EPTPC_TIMER2_RISE | EPTPC TIMER2 RISE. |
| FSP_SIGNAL_EPTPC_TIMER3_FALL | EPTPC TIMER3 FALL. |
| FSP_SIGNAL_EPTPC_TIMER3_RISE | EPTPC TIMER3 RISE. |
| FSP_SIGNAL_EPTPC_TIMER4_FALL | EPTPC TIMER4 FALL. |
| FSP_SIGNAL_EPTPC_TIMER4_RISE | EPTPC TIMER4 RISE. |
| FSP_SIGNAL_EPTPC_TIMER5_FALL | EPTPC TIMER5 FALL. |
| FSP_SIGNAL_EPTPC_TIMER5_RISE | EPTPC TIMER5 RISE. |
| FSP_SIGNAL_FCU_FIFERR | FCU FIFERR. |
| FSP_SIGNAL_FCU_FRDYI | FCU FRDYI. |
| FSP_SIGNAL_GLCDC_LINE_DETECT | GLCDC LINE DETECT. |
| FSP_SIGNAL_GLCDC_UNDERFLOW_1 | GLCDC UNDERFLOW 1. |
| FSP_SIGNAL_GLCDC_UNDERFLOW_2 | GLCDC UNDERFLOW 2. |
| FSP_SIGNAL_GPT_CAPTURE_COMPARE_A | GPT CAPTURE COMPARE A. |
| FSP_SIGNAL_GPT_CAPTURE_COMPARE_B | GPT CAPTURE COMPARE B. |
| FSP_SIGNAL_GPT_COMPARE_C | GPT COMPARE C. |
| FSP_SIGNAL_GPT_COMPARE_D | GPT COMPARE D. |
| FSP_SIGNAL_GPT_COMPARE_E | GPT COMPARE E. |

| | |
|---|---|
| FSP_SIGNAL_GPT_COMPARE_F | GPT COMPARE F. |
| FSP_SIGNAL_GPT_COUNTER_OVERFLOW | GPT COUNTER OVERFLOW. |
| FSP_SIGNAL_GPT_COUNTER_UNDERFLOW | GPT COUNTER UNDERFLOW. |
| FSP_SIGNAL_GPT_AD_TRIG_A | GPT AD TRIG A. |
| FSP_SIGNAL_GPT_AD_TRIG_B | GPT AD TRIG B. |
| FSP_SIGNAL_OPS_UVW_EDGE | OPS UVW EDGE. |
| FSP_SIGNAL_ICU_IRQ0 | ICU IRQ0. |
| FSP_SIGNAL_ICU_IRQ1 | ICU IRQ1. |
| FSP_SIGNAL_ICU_IRQ2 | ICU IRQ2. |
| FSP_SIGNAL_ICU_IRQ3 | ICU IRQ3. |
| FSP_SIGNAL_ICU_IRQ4 | ICU IRQ4. |
| FSP_SIGNAL_ICU_IRQ5 | ICU IRQ5. |
| FSP_SIGNAL_ICU_IRQ6 | ICU IRQ6. |
| FSP_SIGNAL_ICU_IRQ7 | ICU IRQ7. |
| FSP_SIGNAL_ICU_IRQ8 | ICU IRQ8. |
| FSP_SIGNAL_ICU_IRQ9 | ICU IRQ9. |
| FSP_SIGNAL_ICU_IRQ10 | ICU IRQ10. |
| FSP_SIGNAL_ICU_IRQ11 | ICU IRQ11. |
| FSP_SIGNAL_ICU_IRQ12 | ICU IRQ12. |
| FSP_SIGNAL_ICU_IRQ13 | ICU IRQ13. |
| FSP_SIGNAL_ICU_IRQ14 | ICU IRQ14. |
| FSP_SIGNAL_ICU_IRQ15 | ICU IRQ15. |
| FSP_SIGNAL_ICU_SNOOZE_CANCEL | ICU SNOOZE CANCEL. |
| FSP_SIGNAL_IIC_ERI | IIC ERI. |

| | |
|---|---|
| FSP_SIGNAL_IIC_RXI | IIC RXI. |
| FSP_SIGNAL_IIC_TEI | IIC TEI. |
| FSP_SIGNAL_IIC_TXI | IIC TXI. |
| FSP_SIGNAL_IIC_WUI | IIC WUI. |
| FSP_SIGNAL_IOPORT_EVENT_1 | IOPORT EVENT 1. |
| FSP_SIGNAL_IOPORT_EVENT_2 | IOPORT EVENT 2. |
| FSP_SIGNAL_IOPORT_EVENT_3 | IOPORT EVENT 3. |
| FSP_SIGNAL_IOPORT_EVENT_4 | IOPORT EVENT 4. |
| FSP_SIGNAL_IOPORT_EVENT_B | IOPORT EVENT B. |
| FSP_SIGNAL_IOPORT_EVENT_C | IOPORT EVENT C. |
| FSP_SIGNAL_IOPORT_EVENT_D | IOPORT EVENT D. |
| FSP_SIGNAL_IOPORT_EVENT_E | IOPORT EVENT E. |
| FSP_SIGNAL_IWDT_UNDERFLOW | IWDT UNDERFLOW. |
| FSP_SIGNAL_JPEG_JDTI | JPEG JDTI. |
| FSP_SIGNAL_JPEG_JEDI | JPEG JEDI. |
| FSP_SIGNAL_KEY_INT | KEY INT. |
| FSP_SIGNAL_PDC_FRAME_END | PDC FRAME END. |
| FSP_SIGNAL_PDC_INT | PDC INT. |
| FSP_SIGNAL_PDC_RECEIVE_DATA_READY | PDC RECEIVE DATA READY. |
| FSP_SIGNAL_POEG_EVENT | POEG EVENT. |
| FSP_SIGNAL_QSPI_INT | QSPI INT. |
| FSP_SIGNAL_RTC_ALARM | RTC ALARM. |
| FSP_SIGNAL_RTC_PERIOD | RTC PERIOD. |
| FSP_SIGNAL_RTC_CARRY | RTC CARRY. |

| | |
|---|---|
| FSP_SIGNAL_SCE_INTEGRATE_RDRDY | SCE INTEGRATE RDRDY. |
| FSP_SIGNAL_SCE_INTEGRATE_WRRDY | SCE INTEGRATE WRRDY. |
| FSP_SIGNAL_SCE_LONG_PLG | SCE LONG PLG. |
| FSP_SIGNAL_SCE_PROC_BUSY | SCE PROC BUSY. |
| FSP_SIGNAL_SCE_RDRDY_0 | SCE RDRDY 0. |
| FSP_SIGNAL_SCE_RDRDY_1 | SCE RDRDY 1. |
| FSP_SIGNAL_SCE_ROMOK | SCE ROMOK. |
| FSP_SIGNAL_SCE_TEST_BUSY | SCE TEST BUSY. |
| FSP_SIGNAL_SCE_WRRDY_0 | SCE WRRDY 0. |
| FSP_SIGNAL_SCE_WRRDY_1 | SCE WRRDY 1. |
| FSP_SIGNAL_SCE_WRRDY_4 | SCE WRRDY 4. |
| FSP_SIGNAL_SCI_AM | SCI AM. |
| FSP_SIGNAL_SCI_ERI | SCI ERI. |
| FSP_SIGNAL_SCI_RXI | SCI RXI. |
| FSP_SIGNAL_SCI_RXI_OR_ERI | SCI RXI OR ERI. |
| FSP_SIGNAL_SCI_TEI | SCI TEI. |
| FSP_SIGNAL_SCI_TXI | SCI TXI. |
| FSP_SIGNAL_SDADC_ADI | SDADC ADI. |
| FSP_SIGNAL_SDADC_SCANEND | SDADC SCANEND. |
| FSP_SIGNAL_SDADC_CALIEND | SDADC CALIEND. |
| FSP_SIGNAL_SDHIMMC_ACCS | SDHIMMC ACCS. |
| FSP_SIGNAL_SDHIMMC_CARD | SDHIMMC CARD. |
| FSP_SIGNAL_SDHIMMC_DMA_REQ | SDHIMMC DMA REQ. |
| FSP_SIGNAL_SDHIMMC_SDIO | SDHIMMC SDIO. |

| | |
|---|---|
| FSP_SIGNAL_SPI_ERI | SPI ERI. |
| FSP_SIGNAL_SPI_IDLE | SPI IDLE. |
| FSP_SIGNAL_SPI_RXI | SPI RXI. |
| FSP_SIGNAL_SPI_TEI | SPI TEI. |
| FSP_SIGNAL_SPI_TXI | SPI TXI. |
| FSP_SIGNAL_SRC_CONVERSION_END | SRC CONVERSION END. |
| FSP_SIGNAL_SRC_INPUT_FIFO_EMPTY | SRC INPUT FIFO EMPTY. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_FULL | SRC OUTPUT FIFO FULL. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_OVERFLOW | SRC OUTPUT FIFO OVERFLOW. |
| FSP_SIGNAL_SRC_OUTPUT_FIFO_UNDERFLOW | SRC OUTPUT FIFO UNDERFLOW. |
| FSP_SIGNAL_SSI_INT | SSI INT. |
| FSP_SIGNAL_SSI_RXI | SSI RXI. |
| FSP_SIGNAL_SSI_TXI | SSI TXI. |
| FSP_SIGNAL_SSI_TXI_RXI | SSI TXI RXI. |
| FSP_SIGNAL_TRNG_RDREQ | TRNG RDREQ. |
| FSP_SIGNAL_USB_FIFO_0 | USB FIFO 0. |
| FSP_SIGNAL_USB_FIFO_1 | USB FIFO 1. |
| FSP_SIGNAL_USB_INT | USB INT. |
| FSP_SIGNAL_USB_RESUME | USB RESUME. |
| FSP_SIGNAL_USB_USB_INT_RESUME | USB USB INT RESUME. |
| FSP_SIGNAL_WDT_UNDERFLOW | WDT UNDERFLOW. |
| FSP_SIGNAL_ULPT_COMPARE_A | ULPT COMPARE A. |
| FSP_SIGNAL_ULPT_COMPARE_B | ULPT COMPARE B. |
| FSP_SIGNAL_ULPT_INT | ULPT INT. |

## Function Documentation

### ◆ R_FSP_VersionGet()

fsp_err_t R_FSP_VersionGet ( fsp_pack_version_t *const *p_version*)

Get the FSP version based on compile time macros.

**Parameters**

| | | |
|---|---|---|
| [out] | p_version | Memory address to return version information to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Version information stored. |
| FSP_ERR_ASSERTION | The parameter p_version is NULL. |

### ◆ R_BSP_SourceClockHzGet()

uint32_t R_BSP_SourceClockHzGet ( fsp_priv_source_clock_t *clock*)

Gets the frequency of a source clock.

**Parameters**

| | | |
|---|---|---|
| [in] | clock | Pointer to Octaclk setting structure which provides information regarding Octaclk source and divider settings to be applied. |

**Returns**

Frequency of requested clock in Hertz.

### ◆ R_FSP_CurrentIrqGet()

__STATIC_INLINE IRQn_Type R_FSP_CurrentIrqGet ( void )

Return active interrupt vector number value

**Returns**

Active interrupt vector number value

### ◆ R_FSP_SystemClockHzGet()

__STATIC_INLINE uint32_t R_FSP_SystemClockHzGet ( fsp_priv_clock_t *clock*)

Gets the frequency of a system clock.

**Returns**
Frequency of requested clock in Hertz.

### ◆ R_FSP_ClockDividerGet()

__STATIC_INLINE uint32_t R_FSP_ClockDividerGet ( uint32_t *ckdivcr*)

Converts a clock's CKDIVCR register value to a clock divider (Eg: SPICKDIVCR).

**Returns**
Clock Divider

### ◆ R_BSP_UniqueIdGet()

__STATIC_INLINE bsp_unique_id_t const* R_BSP_UniqueIdGet ( void )

Get unique ID for this device.

**Returns**
A pointer to the unique identifier structure

### ◆ R_BSP_PartNumberGet()

__STATIC_INLINE fsp_err_t R_BSP_PartNumberGet ( bsp_part_number_t *const *p_part_number*)

Get part number for this device.

**Parameters**

| | | |
|---|---|---|
| [out] | p_part_number | Memory address to return MCU's part number to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Part number information stored. |
| FSP_ERR_ASSERTION | The parameter p_part_number is NULL. |
| FSP_ERR_NOT_FOUND | An error occurred when retrieving data from part number register. |

### ◆ R_BSP_FlashCacheDisable()

| __STATIC_INLINE void R_BSP_FlashCacheDisable ( void ) |
|---|
| Disables the flash cache. |

### ◆ R_BSP_FlashCacheEnable()

| __STATIC_INLINE void R_BSP_FlashCacheEnable ( void ) |
|---|
| Enables the flash cache. |

### ◆ R_BSP_SecondaryCoreStart()

| __STATIC_INLINE void R_BSP_SecondaryCoreStart ( void ) |
|---|
| Sets the secondary core VTOR and activates the secondary core. |

### ◆ R_BSP_SoftwareDelay()

| void R_BSP_SoftwareDelay ( uint32_t *delay*, bsp_delay_units_t *units* ) |
|---|
| Delay for at least the specified duration in units and return.<br><br>**Parameters** |

| | | |
|---|---|---|
| [in] | delay | The number of 'units' to delay. |
| [in] | units | The 'base' (bsp_delay_units_t) for the units specified. Valid values are: BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS.<br>For example:<br>At 1 MHz one cycle takes 1 microsecond (.000001 seconds).<br>At 12 MHz one cycle takes 1/12 microsecond or 83 nanoseconds.<br>Therefore one run through bsp_prv_software_delay_loop() takes: ~ (83 * BSP_DELAY_LOOP_CYCLES) or 332 ns. A delay of 2 us therefore requires 2000ns/332ns or 6 loops. |

The 'theoretical' maximum delay that may be obtained is determined by a full 32 bit loop count and the system clock rate. @120MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 120000000) = 143 seconds. @32MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 32000000) = 536 seconds

Note that requests for very large delays will be affected by rounding in the calculations and the actual delay achieved may be slightly longer. @32 MHz, for example, a request for 532 seconds will be closer to 536 seconds.

Note also that if the calculations result in a loop_cnt of zero, the bsp_prv_software_delay_loop() function is not called at all. In this case the requested delay is too small (nanoseconds) to be carried out by the loop itself, and the overhead associated with executing the code to just get to this point has certainly satisfied the requested delay.

***Note***

*This function calls bsp_cpu_clock_get() which ultimately calls R_CGC_SystemClockFreqGet() and therefore requires that the BSP has already initialized the CGC (which it does as part of the Sysinit). Care should be taken to ensure this remains the case if in the future this function were to be called as part of the BSP initialization.*
*This function will delay for **at least** the specified duration. Due to overhead in calculating the correct number of loops to delay, very small delay values (generally 1-5 microseconds) may be significantly longer than specified. Approximate overhead for this function is as follows:*

- *CM4: 20-50 cycles*
- *CM33: 10-60 cycles*
- *CM23: 75-200 cycles*

*If more accurate microsecond timing must be performed in software it is recommended to use bsp_prv_software_delay_loop() directly. In this case, use BSP_DELAY_LOOP_CYCLES or BSP_DELAY_LOOPS_CALCULATE() to convert a calculated delay cycle count to a number of software delay loops.*
*Delays may be longer than expected when compiler optimization is turned off.*

**Warning**

The delay will be longer than specified on CM23 devices when the core clock is greater than 32 MHz. Setting BSP_DELAY_LOOP_CYCLES to 6 will improve accuracy at 48 MHz but will result in shorter than expected delays at lower speeds.

### ◆ R_BSP_GroupIrqWrite()

fsp_err_t R_BSP_GroupIrqWrite ( bsp_grp_irq_t *irq*, void(*)(bsp_grp_irq_t irq) *p_callback* )

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | Interrupt for which to register a callback. |
| [in] | p_callback | Pointer to function to call when interrupt occurs. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback registered |
| FSP_ERR_ASSERTION | Callback pointer is NULL |

### ◆ NMI_Handler()

| void NMI_Handler ( void ) |
|---|
| Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception. |

### ◆ R_BSP_IpcSemaphoreTake()

| fsp_err_t R_BSP_IpcSemaphoreTake ( bsp_ipc_semaphore_handle_t const *const *p_semaphore_handle*) |
|---|
| Attempt to take IPC semaphore<br><br>**Parameters**<br>[in] \| p_semaphore_handle \| Semaphore handle corresponding to IPCSEMn to use.<br><br>**Return values**<br>FSP_SUCCESS \| Semaphore successfully taken<br>FSP_ERR_IN_USE \| Semaphore already taken<br>FSP_ERR_IP_CHANNEL_NOT_PRESENT \| Specified semaphore doesn't exist on device |

### ◆ R_BSP_IpcSemaphoreGive()

| fsp_err_t R_BSP_IpcSemaphoreGive ( bsp_ipc_semaphore_handle_t const *const *p_semaphore_handle*) |
|---|
| Give/clear the IPC semaphore<br><br>**Parameters**<br>[in] \| p_semaphore_handle \| Semaphore handle corresponding to IPCSEMn to use.<br><br>**Return values**<br>FSP_SUCCESS \| Semaphore successfully given<br>FSP_ERR_IP_CHANNEL_NOT_PRESENT \| Specified semaphore doesn't exist on device |

### ◆ R_BSP_IpcNmiEnable()

fsp_err_t R_BSP_IpcNmiEnable ( bsp_ipc_nmi_cb_t *p_callback*)

Enable NMI for current core

**Parameters**

| | | |
|---|---|---|
| [in] | p_callback | Pointer to callback to use for NMI. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | NMI request set |

### ◆ R_BSP_IpcNmiRequestSet()

fsp_err_t R_BSP_IpcNmiRequestSet ( void )

Set NMI request for the opposing core

**Return values**

| | |
|---|---|
| FSP_SUCCESS | NMI request set |

### ◆ R_BSP_IrqClearPending()

void R_BSP_IrqClearPending ( IRQn_Type *irq*)

Clear the interrupt status flag (IR) for a given interrupt and clear the NVIC pending interrupt.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | Interrupt for which to clear the IR bit. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_IrqCfg()

void R_BSP_IrqCfg ( IRQn_Type const *irq*, uint32_t *priority*, void * *p_context* )

Sets the interrupt priority and context.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | The IRQ to configure. |
| [in] | priority | NVIC priority of the interrupt |
| [in] | p_context | The interrupt context is a pointer to data required in the ISR. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_IrqEnableNoClear()

void R_BSP_IrqEnableNoClear ( IRQn_Type const *irq*)

Enable the IRQ in the NVIC (Without clearing the pending bit).

**Parameters**

| | | |
|---|---|---|
| [in] | irq | The IRQ to enable. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_IrqEnable()

void R_BSP_IrqEnable ( IRQn_Type const *irq*)

Clears pending interrupts in both ICU and NVIC, then enables the interrupt.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | Interrupt for which to clear the IR bit and enable in the NVIC. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_IrqDisable()

void R_BSP_IrqDisable ( IRQn_Type const *irq*)

Disables interrupts in the NVIC.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | The IRQ to disable in the NVIC. Note that the enums listed for IRQn_Type are only those for the Cortex Processor Exceptions Numbers. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_IrqCfgEnable()

void R_BSP_IrqCfgEnable ( IRQn_Type const *irq*, uint32_t *priority*, void * *p_context* )

Sets the interrupt priority and context, clears pending interrupts, then enables the interrupt.

**Parameters**

| | | |
|---|---|---|
| [in] | irq | Interrupt number. |
| [in] | priority | NVIC priority of the interrupt |
| [in] | p_context | The interrupt context is a pointer to data required in the ISR. |

**Warning**

Do not call this function for system exceptions where the IRQn_Type value is < 0.

### ◆ R_BSP_MemoryMirrorStatusGet()

__STATIC_INLINE fsp_err_t R_BSP_MemoryMirrorStatusGet ( mmf_status_t * *p_mmf_status*)

Get the current status of Memory Mirror.

**Parameters**

| | | |
|---|---|---|
| [out] | p_mmf_status | Pointer to instance which used for storing the state of MMF after invoked this function. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MMF status retrieved successfully. |
| FSP_ERR_UNSUPPORTED | MCU does not support MMF. |
| FSP_ERR_ASSERTION | NULL pointer passed as argument. |

This function retrieves the current state of the MMF and the mirrored address into a user provided structure.

### ◆ R_BSP_MemoryMirrorAddrSet()

__STATIC_INLINE fsp_err_t R_BSP_MemoryMirrorAddrSet ( const uint32_t *addr*)

Set address for MMF region.

**Parameters**

| | | |
|---|---|---|
| [in] | addr | Address of memory region to be mirrored into MMF region. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address is set successfully. |
| FSP_ERR_UNSUPPORTED | MCU does not support MMF. |
| FSP_ERR_INVALID_ADDRESS | Requested address is out of supported range. |

This function sets the memory address to be mirrored by MMF.

### ◆ R_BSP_RegisterProtectEnable()

void R_BSP_RegisterProtectEnable ( bsp_reg_protect_t *regs_to_protect*)

Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

**Parameters**

| | | |
|---|---|---|
| [in] | regs_to_protect | Registers which have write protection enabled. |

### ◆ R_BSP_RegisterProtectDisable()

void R_BSP_RegisterProtectDisable ( bsp_reg_protect_t *regs_to_unprotect*)

Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

**Parameters**

| | | |
|---|---|---|
| [in] | regs_to_unprotect | Registers which have write protection disabled. |

### ◆ Reset_Handler()

| void Reset_Handler ( void ) |
|---|
| MCU starts executing here out of reset. Main stack pointer is set up already. |

### ◆ Default_Handler()

| void Default_Handler ( void ) |
|---|
| Default exception handler. |

### ◆ SystemInit()

| void SystemInit ( void ) |
|---|
| Initialize the MCU and the runtime environment. |

### ◆ R_BSP_WarmStart()

| void R_BSP_WarmStart ( bsp_warm_start_event_t *event*) |
|---|
| This function is called at various points during the startup process. This function is declared as a weak symbol higher up in this file because it is meant to be overridden by a user implemented version. One of the main uses for this function is to call functional safety code during the startup process. To use this function just copy this function into your own code and modify it to meet your needs.<br><br>**Parameters**<br>[in] event — Where the code currently is in the start up process |

## Variable Documentation

### ◆ BSP_SECTION_EARLY_INIT

| uint32_t SystemCoreClock BSP_SECTION_EARLY_INIT |
|---|
| System Clock Frequency (Core Clock) |

#### 5.1.5.1 RA0E1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra0e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillation Stabilization Time | • 2^8/MOSC<br>• 2^9/MOSC<br>• 2^10/MOSC<br>• 2^11/MOSC<br>• 2^13/MOSC<br>• 2^15/MOSC<br>• 2^17/MOSC<br>• 2^18/MOSC | 2^18/MOSC | |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Fixed vector enumeration<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.2 RA0E2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra0e2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillation Stabilization Time | • 2^8/MOSC<br>• 2^9/MOSC<br>• 2^10/MOSC | 2^18/MOSC | |

- 2^11/MOSC
- 2^13/MOSC
- 2^15/MOSC
- 2^17/MOSC
- 2^18/MOSC

| | | | |
|---|---|---|---|
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Fixed vector enumeration<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.3 RA0L1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra0l1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillation Stabilization Time | • 2^8/MOSC<br>• 2^9/MOSC<br>• 2^10/MOSC<br>• 2^11/MOSC<br>• 2^13/MOSC<br>• 2^15/MOSC<br>• 2^17/MOSC<br>• 2^18/MOSC | 2^18/MOSC | |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz |

and requires all clock dividers to be at least 4 when oscillation stop detection is used.

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list is device specific.*

### ◆ icu_event_t

enum icu_event_t

Fixed vector enumeration

***Note***
*This list is device specific.*

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

#### 5.1.5.4 RA2A1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra2a1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | • Enabled<br>• Disabled | Disabled | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

#### Macros

| | |
| --- | --- |
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.5 RA2A2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra2a2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but |

| | | | |
|---|---|---|---|
| | | | will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Events to be used with the IELSR register to link interrupt events to the NVIC<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.6 RA2E1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra2e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

- 8192 cycles
- 16384 cycles
- 32768 cycles
- 65536 cycles
- 131072 cycles
- 262144 cycles

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

***Note***

*This list is device specific.*

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

### 5.1.5.7 RA2E2

BSP » MCU Board Support Package

#### Detailed Description

**Build Time Configurations for ra2e2_fsp**

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |

#### Macros

| | |
| --- | --- |
| #define | BSP_ELC_PERIPHERAL_MASK |

#### Enumerations

| | |
| --- | --- |
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

#### Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ icu_event_t

| enum icu_event_t |
|---|
| Events to be used with the IELSR register to link interrupt events to the NVIC<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.8 RA2E3

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra2e3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Events to be used with the IELSR register to link interrupt events to the NVIC<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.9 RA2L1

BSP » MCU Board Support Package

### Functions

| | |
| --- | --- |
| bsp_power_mode_t | R_BSP_PowerModeSet (bsp_power_mode_t mode) |

### Detailed Description

#### Build Time Configurations for ra2l1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Power | | | |
| DC-DC Regulator | • Disabled<br>• Enabled<br>• Enabled at startup | Disabled | To use the DCDC regulator an external inductor and capacitor must be connected as |

| | | | |
|---|---|---|---|
| | | | specified in chapter 40 of the RA2L1 manual. In addition the supply voltage must be above 2.4V and ICLK must be 2 MHz or higher.<br><br>When set to 'Enabled at startup' the BSP will switch to the DCDC regulator during startup using the voltage range specified below. |
| DC-DC Supply Range | • 2.4V to 2.7V<br>• 2.7V to 3.6V<br>• 3.6V to 4.5V<br>• 4.5V to 5.5V | 2.7V to 3.6V | Set this to the expected MCU supply voltage (Vcc) at startup when using the DCDC regulator. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file.

## Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

## Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |
| enum | bsp_power_mode_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list is device specific.*

### ◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

***Note***
*This list is device specific.*

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***
*This list is device specific.*

### ◆ bsp_power_mode_t

| enum bsp_power_mode_t | |
|---|---|
| Voltage regulator mode | |
| Enumerator | |
| BSP_POWER_MODE_DCDC_2V4_TO_2V7 | DCDC mode; 2.4V to 2.7V supply. |
| BSP_POWER_MODE_DCDC_2V7_TO_3V6 | DCDC mode; 2.7V to 3.6V supply. |
| BSP_POWER_MODE_DCDC_3V6_TO_4V5 | DCDC mode; 3.6V to 4.5V supply. |
| BSP_POWER_MODE_DCDC_4V5_TO_5V5 | DCDC mode; 4.5V to 5.5V supply. |
| BSP_POWER_MODE_LDO | LDO mode. |

## Function Documentation

### ◆ R_BSP_PowerModeSet()

bsp_power_mode_t R_BSP_PowerModeSet ( bsp_power_mode_t *mode*)

Select either the LDO or DCDC regulator and/or update the MCU supply voltage range. Returns the previously selected mode.

***Note***

*DCDC mode has the following limitations:*

- *Supply voltage must be 2.4V or greater*
- *Low- and Subosc-speed modes are not available*
- *Software Standby is not available Ensure these limitations are respected before entering DCDC mode. If supply voltage may drop below 2.4V during operation, configure a LVD channel to interrupt or reset the MCU near this threshold to switch back to the LDO.*

*Switching to DCDC mode temporarily disables all interrupts and blocks for 22 microseconds; switching to LDO from DCDC temporarily disables all peripherals and interrupts and blocks for 60 microseconds.*

*If the supply voltage falls outside the range originally specified when starting the DCDC regulator, call this function again with the updated supply voltage.*

**Returns**

The previously selected power mode.

## 5.1.5.10 RA2L2

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra2l2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Events to be used with the IELSR register to link interrupt events to the NVIC<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.11 RA2T1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra2t1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Disabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | Not Supported | Not Supported | Use the low voltage mode. This limits the ICLK operating |

| | | | |
|---|---|---|---|
| | | | frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | icu_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ icu_event_t

| enum icu_event_t |
| --- |
| Events to be used with the IELSR register to link interrupt events to the NVIC<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.12 RA4C1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra4c1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines |

| | | | |
|---|---|---|---|
| | | | whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |

| | | | |
|---|---|---|---|
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with |

| | | | |
|---|---|---|---|
| | | | TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | Disabled | Disabled | FLL is not supported. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

## Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

## Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.13 RA4E1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra4e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone |

| | | | |
|---|---|---|---|
| | | | Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are | config.bsp.fsp.tz.stbra | Defines whether |

| | | | |
|---|---|---|---|
| | all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | msar.both | Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |

| | | | |
|---|---|---|---|
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue |

| | | | |
|---|---|---|---|
| | | | where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

## Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| |
|---|
| enum elc_event_t |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| |
|---|
| enum elc_peripheral_t |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.14 RA4E2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra4e2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |

| | | | |
|---|---|---|---|
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non- |

| | | | |
|---|---|---|---|
| | • Disable Uninitialized Non-Secure Application Fallback | | secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait | • 3 cycles | 8163 cycles | Number of cycles to |

| | | |
|---|---|---|
| Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | wait for the main oscillator clock to stabilize. |

**Macros**

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

**Enumerations**

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.15 RA4L1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra4l1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non- |

| | | | |
|---|---|---|---|
| | | | secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure | Secure State | Value for SCB->AIRCR register bit |

| | | | |
|---|---|---|---|
| | State<br>• Secure State | | SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | Disabled | Disabled | FLL is not supported. |
| Startup C-Cache Line | • 32 Bytes | 32 Bytes | Set the C-Cache line |

| | | | |
|---|---|---|---|
| Size | • 64 Bytes | | size configured during startup. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***

*This list is device specific.*

### 5.1.5.16 RA4M1

BSP » MCU Board Support Package

#### Detailed Description

#### Build Time Configurations for ra4m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | • Enabled<br>• Disabled | Disabled | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles<br>• 8192 cycles<br>• 16384 cycles<br>• 32768 cycles<br>• 65536 cycles<br>• 131072 cycles<br>• 262144 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

#### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

#### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

#### Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
| --- |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.17 RA4M2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra4m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non- |

| | | | |
|---|---|---|---|
| | • Secure State | | secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers |

| | | | |
|---|---|---|---|
| | • Secure State | | are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non- |

| | | | |
|---|---|---|---|
| | • Disable Uninitialized Non-Secure Application Fallback | | secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait | • 3 cycles | 8163 cycles | Number of cycles to |

| | | |
|---|---|---|
| Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | wait for the main oscillator clock to stabilize. |

**Macros**

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

**Enumerations**

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.18 RA4M3

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra4m3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non- |

| | | | |
|---|---|---|---|
| | | | secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |

| | | | |
|---|---|---|---|
| | Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | | |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure | Both Secure and Non-Secure State | Defines whether the reset status registers |

| | | | |
|---|---|---|---|
| | State<br>• Secure State | | (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. |

| | | | |
|---|---|---|---|
| | | | The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.19 RA4T1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra4t1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non- |

| | | | |
|---|---|---|---|
| | • Secure State | | secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status | • Both Secure | Both Secure and Non- | Defines whether the |

| | | | |
|---|---|---|---|
| Accessibility | and Non-Secure State<br>• Secure State | Secure State | reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and |

| | | | |
|---|---|---|---|
| | | | Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| TFU Mathlib | • Disabled<br>• Enabled | Enabled | If enabled, trigonometric library functions sinf, cosf, atan2f, and hypotf are replaced with hardware accelerated TFU functions. Disable this if reentrant access to these functions is required. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.20 RA4W1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra4w1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Use Low Voltage Mode | • Enabled<br>• Disabled | Disabled | Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4. |
| Main Oscillator Wait Time | • 2 cycles<br>• 1024 cycles<br>• 2048 cycles<br>• 4096 cycles | 262144 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

- 8192 cycles
- 16384 cycles
- 32768 cycles
- 65536 cycles
- 131072 cycles
- 262144 cycles

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

### Enumeration Type Documentation

#### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list is device specific.*

#### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***
*This list is device specific.*

## 5.1.5.21 RA6E1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra6e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only |

| | | | |
|---|---|---|---|
| | | | valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all |

| | | | |
|---|---|---|---|
| | | | battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. |

| | | | |
|---|---|---|---|
| | | | When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.22 RA6E2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6e2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and |

| | | | |
|---|---|---|---|
| | | | NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non- | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the |

| | | | |
|---|---|---|---|
| | secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | | Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request | • Both Secure | Secure State | Value for SCB->AIRCR |

| | | | |
|---|---|---|---|
| Accessibility | and Non-Secure State<br>• Secure State | | register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves |

| | | | |
|---|---|---|---|
| | | | HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.23 RA6M1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain |

| | | | |
|---|---|---|---|
| | | | restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| MOSC Drive Capability Auto Switching | • Enabled<br>• Disabled | Disabled | When enabled, the drive capability auto switching function lowers the MOSC drive capability automatically after the MOSC starts and suppresses the EMI associated with the MOSC. |
| Main Oscillator Wait Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |

enum elc_peripheral_t

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.24 RA6M2

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| SDRAM | | | |
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles | 6 cycles | Row Active Interval |

| | | | |
|---|---|---|---|
| | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | | |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles | 3 cycles | Column Latency |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles<br>• 9 cycles<br>• 10 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and |

| | | | |
|---|---|---|---|
| | | | configured by the BSP with the provided settings. |
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit | 16-bit | Specifies the data bus width for SDRAM. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| MOSC Drive Capability Auto Switching | • Enabled<br>• Disabled | Disabled | When enabled, the drive capability auto switching function lowers the MOSC drive |

| | | | |
|---|---|---|---|
| | | | capability automatically after the MOSC starts and suppresses the EMI associated with the MOSC. |
| Main Oscillator Wait Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

### 5.1.5.25 RA6M3

BSP » MCU Board Support Package

#### Detailed Description

**Build Time Configurations for ra6m3_fsp**

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| SDRAM | | | |
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | 6 cycles | Row Active Interval |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles | 3 cycles | Column Latency |

| | | | |
|---|---|---|---|
| | • 3 cycles | | |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles<br>• 9 cycles<br>• 10 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and configured by the BSP with the provided settings. |
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit | 16-bit | Specifies the data bus width for SDRAM. |
| Clocks | | | |

| | | | |
|---|---|---|---|
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| MOSC Drive Capability Auto Switching | • Enabled<br>• Disabled | Disabled | When enabled, the drive capability auto switching function lowers the MOSC drive capability automatically after the MOSC starts and suppresses the EMI associated with the MOSC. |
| Main Oscillator Wait Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |

**Macros**

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.26 RA6M4

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m4_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write |

| | | | |
|---|---|---|---|
| | State<br>• Secure State | | accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are |

| | | | |
|---|---|---|---|
| | | | write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible |

| | | | |
|---|---|---|---|
| | • Secure State | | for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time |

| | | | |
|---|---|---|---|
| | | | when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.27 RA6M5

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra6m5_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
| --- | --- | --- | --- |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and |

| | | | |
|---|---|---|---|
| | | | NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non- | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the |

| | | | |
|---|---|---|---|
| | secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | | Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request | • Both Secure | Secure State | Value for SCB->AIRCR |

| | | | |
|---|---|---|---|
| Accessibility | and Non-Secure State<br>• Secure State | | register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non- |

| | | | |
|---|---|---|---|
| | • Disable Uninitialized Non-Secure Application Fallback | | secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| Main Oscillator Wait | • 3 cycles | 8163 cycles | Number of cycles to |

| | | |
|---|---|---|
| Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | wait for the main oscillator clock to stabilize. |

**Macros**

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

**Enumerations**

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.28 RA6T1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra6t1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| MOSC Drive Capability Auto Switching | • Enabled<br>• Disabled | Disabled | When enabled, the drive capability auto switching function lowers the MOSC drive capability automatically after the MOSC starts and suppresses the EMI associated with the MOSC. |
| Main Oscillator Wait Time | • 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

| | | | |
|---|---|---|---|
| | • 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | | |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***
*This list is device specific.*

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***
*This list is device specific.*

### 5.1.5.29 RA6T2

BSP » MCU Board Support Package

#### Detailed Description

#### Build Time Configurations for ra6t2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less |

| | | | |
|---|---|---|---|
| | | | bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby RAM | • Regions 7-0 are all Secure.<br>• Region 7 is Non-secure. Regions 6-0 are Secure.<br>• Regions 7-6 are Non-secure. Regions 5-0 are Secure.<br>• Regions 7-5 are Non-secure. Regions 4-0 are Secure.<br>• Regions 7-4 are Non-secure. Regions 3-0 are Secure.<br>• Regions 7-3 are Non-secure. Regions 2-0 are Secure.<br>• Regions 7-2 are Non-secure. | config.bsp.fsp.tz.stbramsar.both | Defines whether Standby RAM registers are accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |

| | | | |
|---|---|---|---|
| | Regions 1-0 are Secure.<br>• Regions 7-1 are Non-secure. Region 0 is Secure.<br>• Regions 7-0 are all Non-secure. | | |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |

| | | | |
|---|---|---|---|
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| TFU Mathlib | • Disabled<br>• Enabled | Enabled | If enabled, trigonometric library functions sinf, cosf, atan2f, and hypotf are replaced with hardware accelerated TFU functions. Disable this if reentrant access to these functions is required. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

- 259 cycles
- 547 cycles
- 1059 cycles
- 2147 cycles
- 4291 cycles
- 8163 cycles

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.30 RA6T3

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra6t3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only |

| | | | |
|---|---|---|---|
| | | | valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMPRCR is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM ECC | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAM ECC registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non- |

| | | | |
|---|---|---|---|
| | | | secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| Cache Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Cache registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. |

| | | | |
|---|---|---|---|
| | | | The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Startup C-Cache Line Size | • 32 Bytes<br>• 64 Bytes | 32 Bytes | Set the C-Cache line size configured during startup. |
| TFU Mathlib | • Disabled<br>• Enabled | Enabled | If enabled, trigonometric library functions sinf, cosf, atan2f, and hypotf are replaced with hardware accelerated TFU functions. Disable this if reentrant access to these functions is required. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| |
|---|
| enum elc_event_t |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| |
|---|
| enum elc_peripheral_t |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.31 RA8D1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra8d1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| SDRAM | | | |

| | | | |
|---|---|---|---|
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | 6 cycles | Row Active Interval |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles | 3 cycles | Column Latency |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles<br>• 9 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |

| | | | |
|---|---|---|---|
| | • 10 cycles | | |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and configured by the BSP with the provided settings. |
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit<br>• 32-bit | 16-bit | Specifies the data bus width for SDRAM. |
| OSPI_B | | | |
| Startup Support | • Enabled<br>• Disabled | Disabled | If enabled, the OSPI_B driver will be initialized and configured by the BSP. A 'Storage > OSPI (r_ospi_b)' or 'Storage > SiP Flash (r_ospi_b)' stack must be configured. |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State | Secure State | Value for SCB->AIRCR register bit |

| | | | |
|---|---|---|---|
| | • Secure State | | BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM0 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM1 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR1, and SRAMESCLR.CLR1 are write accessible for the Non-secure application. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| Standby SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether STBRAMCR, and SRAMESCLR.CLRS are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register C | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the SDRAM/CSC Control registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the |

| | | | |
|---|---|---|---|
| | | | SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Graphics Power Domain Security Attribution | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Defines whether the PDCTRGD register is write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has |

| | | | |
|---|---|---|---|
| | Application Fallback | | not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Clock Settling Delay | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RA8D1 HWM (R01UH0995EJ0100) section 8.11.1 for details. |
| Sleep Mode Entry and Exit Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays before and after |

| | | | |
|---|---|---|---|
| | | | entering sleep modes to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 100MHz. See RA8D1 HWM (R01UH0995EJ0100) Section 10.8.10 for details. |
| RTOS Sleep on Idle | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled will allow RTOS ports to enter CPU Sleep mode while idle. This should not be used when CPUCLK is configured over 120MHz. See RA8M1 HWM (R01UH0994EJ0100) Section 10.7.10 for details. |
| MSTP Change Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays after setting a MSTP bit to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8D1 HWM (R01UH0995EJ0100) Section 10.4 for details. |
| Settling Delay (us) | Unit must be a non-negative integer | 150 | Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays. |
| Cache settings | | | |
| Data cache | • Enabled<br>• Disabled | Disabled | Enable limited D-Cache support. See BSP usage notes for limitations. |
| I/O Ports | | | |
| I/O Ports > Voltage Mode | | | |
| VCC | • Low (VCC < 2.7 V)<br>• High (VCC >= 2.7 V) | High (VCC >= 2.7 V) | When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low. |

| | | | |
|---|---|---|---|
| VCC2 | • Low (VCC2 < 2.7 V)<br>• High (VCC2 >= 2.7 V) | High (VCC2 >= 2.7 V) | When VCC2 is lower than 2.7 V, set VCC2 I/O Ports Voltage Mode to Low. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |
| enum | bsp_clocks_octaclk_div_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| |
|---|
| enum elc_event_t |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

### ◆ bsp_clocks_octaclk_div_t

| enum bsp_clocks_octaclk_div_t | |
| --- | --- |
| OCTACLK divider values. | |
| Enumerator | |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_10 | Divide OCTA source clock by 10. |
| BSP_CLOCKS_OCTACLK_DIV_16 | Divide OCTA source clock by 16. |
| BSP_CLOCKS_OCTACLK_DIV_32 | Divide OCTA source clock by 32. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |

| | |
|---|---|
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |

### 5.1.5.32 RA8D2

BSP » MCU Board Support Package

#### Detailed Description

#### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

#### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

#### Macro Definition Documentation

◆ **BSP_ELC_PERIPHERAL_MASK**

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

#### Enumeration Type Documentation

◆ **elc_event_t**

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

◆ **elc_peripheral_t**

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***

*This list is device specific.*

### 5.1.5.33 RA8E1

BSP » MCU Board Support Package

## Detailed Description

### Build Time Configurations for ra8e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| OSPI_B | | | |
| Startup Support | • Enabled<br>• Disabled | Disabled | If enabled, the OSPI_B driver will be initialized and configured by the BSP. A 'Storage > OSPI (r_ospi_b)' or 'Storage > SiP Flash (r_ospi_b)' stack must be configured. |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then |

| | | | |
|---|---|---|---|
| | | | setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM1 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR1, and SRAMESCLR.CLR1 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether STBRAMCR, and SRAMESCLR.CLRS are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure |

| | | | |
|---|---|---|---|
| | | | application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non- |

| | | | |
|---|---|---|---|
| | Non-Secure Application Fallback | | secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Clock Settling Delay | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RA8E1 HWM (R01UH1129EJ0100) section 8.11.1 for details. |
| Sleep Mode Entry and Exit Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert |

| | | | |
|---|---|---|---|
| | | | delays before and after entering sleep modes to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8E1 HWM (R01UH1129EJ0100) Section 10.8.10 for details. |
| RTOS Sleep on Idle | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled will allow RTOS ports to enter CPU Sleep mode while idle. This should not be used when CPUCLK is configured over 120MHz. See RA8E1 HWM (R01UH1129EJ0100) Section 10.8.10 for details. |
| MSTP Change Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays after setting a MSTP bit to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8E1 HWM (R01UH1129EJ0100) Section 10.4 for details. |
| Settling Delay (us) | Unit must be a non-negative integer | 150 | Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays. |
| Cache settings | | | |
| Data cache | • Enabled<br>• Disabled | Disabled | Enable limited D-Cache support. See BSP usage notes for limitations. |
| I/O Ports | | | |
| I/O Ports > Voltage Mode | | | |
| VCC | • Low (VCC < 2.7 V)<br>• High (VCC >= 2.7 V) | High (VCC >= 2.7 V) | When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low. |

| | | | |
|---|---|---|---|
| VCC2 | • Low (VCC2 < 2.7 V)<br>• High (VCC2 >= 2.7 V) | High (VCC2 >= 2.7 V) | When VCC2 is lower than 2.7 V, set VCC2 I/O Ports Voltage Mode to Low. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |
| enum | bsp_clocks_octaclk_div_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

| |
|---|
| enum elc_event_t |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***

*This list is device specific.*

### ◆ bsp_clocks_octaclk_div_t

enum bsp_clocks_octaclk_div_t

OCTACLK divider values.

| Enumerator | |
|---|---|
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_10 | Divide OCTA source clock by 10. |
| BSP_CLOCKS_OCTACLK_DIV_16 | Divide OCTA source clock by 16. |
| BSP_CLOCKS_OCTACLK_DIV_32 | Divide OCTA source clock by 32. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |

| | |
|---|---|
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |

### 5.1.5.34 RA8E2

BSP » MCU Board Support Package

#### Detailed Description

#### Build Time Configurations for ra8e2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| SDRAM | | | |
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | 6 cycles | Row Active Interval |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles | 3 cycles | Column Latency |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |

| | | | |
|---|---|---|---|
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles<br>• 9 cycles<br>• 10 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and configured by the BSP with the provided settings. |
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit<br>• 32-bit | 16-bit | Specifies the data bus width for SDRAM. |
| OSPI_B | | | |
| Startup Support | • Enabled<br>• Disabled | Disabled | If enabled, the OSPI_B driver will be initialized and configured by the BSP. A 'Storage > OSPI (r_ospi_b)' or 'Storage > SiP Flash (r_ospi_b)' stack must be configured. |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt | Non-Maskable Interrupt | Configure the result of a TrustZone Filter |

| | | | |
|---|---|---|---|
| | • Reset | | exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM0 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01 are write accessible for the Non-secure |

| | | | |
|---|---|---|---|
| | | | application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM1 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR1, and SRAMESCLR.CLR1 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether STBRAMCR, and SRAMESCLR.CLRS are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register C | • Both Secure and Non-Secure | Both Secure and Non-Secure State | Defines whether the SDRAM/CSC Control |

| | | | |
|---|---|---|---|
| | State<br>• Secure State | | registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Graphics Power Domain Security Attribution | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Defines whether the PDCTRGD register is write accessible for the Non-secure application. |

| | | | |
|---|---|---|---|
| | | | This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Clock Settling Delay | • Enabled | Enabled | Setting this option to |

| | | | |
|---|---|---|---|
| | • Disabled | | Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RA8E2 HWM (R01UH1130EJ0100) section 8.11.1 for details. |
| Sleep Mode Entry and Exit Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays before and after entering sleep modes to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 100MHz. See RA8E2 HWM (R01UH1130EJ0100) Section 10.8.10 for details. |
| RTOS Sleep on Idle | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled will allow RTOS ports to enter CPU Sleep mode while idle. This should not be used when CPUCLK is configured over 120MHz. See RA8E2 HWM (R01UH1130EJ0100) Section 10.8.10 for details. |
| MSTP Change Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays after setting a MSTP bit to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8E2 HWM (R01UH1130EJ0100) Section 10.4 for details. |
| Settling Delay (us) | Unit must be a non-negative integer | 150 | Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays. |
| Cache settings | | | |

| | | | |
|---|---|---|---|
| Data cache | • Enabled<br>• Disabled | Disabled | Enable limited D-Cache support. See BSP usage notes for limitations. |
| I/O Ports | | | |
| I/O Ports > Voltage Mode | | | |
| VCC | • Low (VCC < 2.7 V)<br>• High (VCC >= 2.7 V) | High (VCC >= 2.7 V) | When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low. |
| VCC2 | • Low (VCC2 < 2.7 V)<br>• High (VCC2 >= 2.7 V) | High (VCC2 >= 2.7 V) | When VCC2 is lower than 2.7 V, set VCC2 I/O Ports Voltage Mode to Low. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

## Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

## Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |
| enum | bsp_clocks_octaclk_div_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| |
|---|
| #define BSP_ELC_PERIPHERAL_MASK |
| Positions of event link set registers (ELSRs) available on this MCU |

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***

*This list is device specific.*

### ◆ bsp_clocks_octaclk_div_t

| enum bsp_clocks_octaclk_div_t | |
|---|---|
| OCTACLK divider values. | |
| Enumerator | |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_10 | Divide OCTA source clock by 10. |
| BSP_CLOCKS_OCTACLK_DIV_16 | Divide OCTA source clock by 16. |
| BSP_CLOCKS_OCTACLK_DIV_32 | Divide OCTA source clock by 32. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |

| | |
|---|---|
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |

### 5.1.5.35 RA8M1

BSP » MCU Board Support Package

#### Detailed Description

#### Build Time Configurations for ra8m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| SDRAM | | | |
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | 6 cycles | Row Active Interval |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles | 3 cycles | Column Latency |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |

| | | | |
|---|---|---|---|
| | • 8 cycles<br>• 9 cycles<br>• 10 cycles | | |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and configured by the BSP with the provided settings. |
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit<br>• 32-bit | 16-bit | Specifies the data bus width for SDRAM. |
| OSPI_B | | | |
| Startup Support | • Enabled<br>• Disabled | Disabled | If enabled, the OSPI_B driver will be initialized and configured by the BSP. A 'Storage > OSPI (r_ospi_b)' or 'Storage > SiP Flash (r_ospi_b)' stack must be configured. |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |

| | | | |
|---|---|---|---|
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM0 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM1 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR1, and SRAMESCLR.CLR1 are write accessible for the |

| | | | |
|---|---|---|---|
| | | | Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether STBRAMCR, and SRAMESCLR.CLRS are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution Register A | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register C | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the SDRAM/CSC Control registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure | Secure State | Value for SCB->AIRCR register bit |

| | | | |
|---|---|---|---|
| | State<br>• Secure State | | SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the |

| | | | |
|---|---|---|---|
| | | | MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Clock Settling Delay | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RA8M1 HWM (R01UH0994EJ0100) section 8.11.1 for details. |
| Sleep Mode Entry and Exit Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays before and after entering sleep modes to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8M1 HWM (R01UH0994EJ0100) |

| | | | |
|---|---|---|---|
| | | | Section 10.7.10 for details. |
| RTOS Sleep on Idle | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled will allow RTOS ports to enter CPU Sleep mode while idle. This should not be used when CPUCLK is configured over 120MHz. See RA8M1 HWM (R01UH0994EJ0100) Section 10.7.10 for details. |
| MSTP Change Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays after setting a MSTP bit to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8M1 HWM (R01UH0994EJ0100) Section 10.4 for details. |
| Settling Delay (us) | Unit must be a non-negative integer | 150 | Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays. |
| Cache settings | | | |
| Data cache | • Enabled<br>• Disabled | Disabled | Enable limited D-Cache support. See BSP usage notes for limitations. |
| I/O Ports | | | |
| I/O Ports > Voltage Mode | | | |
| VCC | • Low (VCC < 2.7 V)<br>• High (VCC >= 2.7 V) | High (VCC >= 2.7 V) | When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low. |
| VCC2 | • Low (VCC2 < 2.7 V)<br>• High (VCC2 >= 2.7 V) | High (VCC2 >= 2.7 V) | When VCC2 is lower than 2.7 V, set VCC2 I/O Ports Voltage Mode to Low. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease |

| | | | |
|---|---|---|---|
| | | | cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |
| enum | bsp_clocks_octaclk_div_t |

## Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

#define BSP_ELC_PERIPHERAL_MASK

Positions of event link set registers (ELSRs) available on this MCU

## Enumeration Type Documentation

### ◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

***Note***

*This list is device specific.*

### ◆ elc_peripheral_t

enum elc_peripheral_t

Possible peripherals to be linked to event signals

***Note***

*This list is device specific.*

### ◆ bsp_clocks_octaclk_div_t

| enum bsp_clocks_octaclk_div_t | |
|---|---|
| OCTACLK divider values. | |
| Enumerator | |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_10 | Divide OCTA source clock by 10. |
| BSP_CLOCKS_OCTACLK_DIV_16 | Divide OCTA source clock by 16. |
| BSP_CLOCKS_OCTACLK_DIV_32 | Divide OCTA source clock by 32. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |

| | |
|---|---|
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |
| BSP_CLOCKS_OCTACLK_DIV_1 | Divide OCTA source clock by 1. |
| BSP_CLOCKS_OCTACLK_DIV_2 | Divide OCTA source clock by 2. |
| BSP_CLOCKS_OCTACLK_DIV_4 | Divide OCTA source clock by 4. |
| BSP_CLOCKS_OCTACLK_DIV_6 | Divide OCTA source clock by 6. |
| BSP_CLOCKS_OCTACLK_DIV_8 | Divide OCTA source clock by 8. |
| BSP_CLOCKS_OCTACLK_DIV_3 | Divide OCTA source clock by 3. |
| BSP_CLOCKS_OCTACLK_DIV_5 | Divide OCTA source clock by 5. |

### 5.1.5.36 RA8M2

BSP » MCU Board Support Package

#### Detailed Description

#### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

#### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.37 RA8P1

BSP » MCU Board Support Package

### Detailed Description

#### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

#### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
|---|
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
|---|
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.38 RA8T1

BSP » MCU Board Support Package

### Detailed Description

#### Build Time Configurations for ra8t1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| SDRAM | | | |
| SDRAM > Timings | | | |
| tRAS (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles | 6 cycles | Row Active Interval |

| | | | |
|---|---|---|---|
| | • 6 cycles<br>• 7 cycles | | |
| tRCD (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles | 3 cycles | Row Column Latency |
| tRP (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles | 3 cycles | Row Precharge Interval |
| tWR (cycles) | • 1 cycles<br>• 2 cycles | 2 cycles | Write Recovery Interval |
| tCL (cycles) | • 1 cycles<br>• 2 cycles<br>• 3 cycles | 3 cycles | Column Latency |
| tRFC (cycles) | tRFC must be between 2 and 4096 cycles | 937 | Auto-Refresh Request Interval Setting |
| tREFW (cycles) | Refer to the RA Configuration tool for available options. | 8 cycles | Auto-Refresh Cycle/Self-Refresh Clearing Cycle Count Setting. |
| SDRAM > Initialization | | | |
| Auto-Refresh Interval (ARFI) | Refer to the RA Configuration tool for available options. | 10 cycles | Specifies the interval at which the auto-refresh commands are issued in the SDRAM initialization sequence. |
| Auto-Refresh Count (ARFC) | Refer to the RA Configuration tool for available options. | 8 times | Specifies the number of times auto-refresh is to be performed in the SDRAM initialization sequence. |
| Precharge Cycle Count (PRC) | • 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles<br>• 8 cycles<br>• 9 cycles<br>• 10 cycles | 3 cycles | Specifies the number of precharged cycles in the SDRAM initialization sequence. |
| SDRAM Support | • Enabled<br>• Disabled | Disabled | If enabled, SDRAM will be initialized and configured by the BSP with the provided settings. |

| | | | |
|---|---|---|---|
| Address Multiplex Shift | • 8-bit shift<br>• 9-bit shift<br>• 10-bit shift<br>• 11-bit shift | 9-bit shift | Selects the size of the shift towards the lower half of the row address in row address/column address multiplexing. |
| Endian Mode | • Little Endian<br>• Big Endian | Little Endian | Specifies the endianness of the SDRAM address space. See HWM for full list of constraints when using Big Endian. |
| Continuous Access Mode | • Enabled<br>• Disabled | Enabled | If enabled, SDRAM continuous access mode will be enabled. |
| Bus Width | • 8-bit<br>• 16-bit<br>• 32-bit | 16-bit | Specifies the data bus width for SDRAM. |
| Security | | | |
| Security > Exceptions | | | |
| Exception Response | • Non-Maskable Interrupt<br>• Reset | Non-Maskable Interrupt | Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region.<br><br>This setting is only valid when building projects with TrustZone. |
| BusFault, HardFault, and NMI Target | • Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception.<br><br>This setting is only valid when building projects with TrustZone. |
| Prioritize Secure Exceptions | • Enabled<br>• Disabled | Disabled | Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted |

| | | | |
|---|---|---|---|
| | | | by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > SRAM Accessibility | | | |
| SRAM0 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR0, SRAMECCRGN0, SRAMESCLR.CLR00, and SRAMESCLR.CLR01 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| SRAM1 Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether SRAMCR1, and SRAMESCLR.CLR1 are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Standby SRAM Protection | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether STBRAMCR, and SRAMESCLR.CLRS are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Security > BUS Accessibility | | | |
| Bus Security Attribution | • Both Secure | Both Secure and Non- | Defines whether the |

| | | | |
|---|---|---|---|
| Register A | and Non-Secure State<br>• Secure State | Secure State | Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register B | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| Bus Security Attribution Register C | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the SDRAM/CSC Control registers are write accessible for the Non-secure application.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Request Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Secure State | Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use.<br><br>This setting is only valid when building projects with TrustZone. |
| System Reset Status Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.<br><br>This setting is only valid when building projects with |

| | | | |
|---|---|---|---|
| | | | TrustZone. |
| Battery Backup Accessibility | • Both Secure and Non-Secure State<br>• Secure State | Both Secure and Non-Secure State | Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only.<br><br>This setting is only valid when building projects with TrustZone. |
| Uninitialized Non-Secure Application Fallback | • Enable Uninitialized Non-Secure Application Fallback<br>• Disable Uninitialized Non-Secure Application Fallback | Enable Uninitialized Non-Secure Application Fallback | If enabled, the secure application checks if the non-secure application has been programmed in non-secure flash before branching. If the non-secure application has not been programmed, then the secure application branches to an infinite loop in non-secure RAM. This prevents an issue where the debugger may not connect if the MCU is configured in the NSECSD lifecycle state. |
| Clocks | | | |
| HOCO FLL Function | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.<br><br>The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL |

| | | | |
|---|---|---|---|
| | | | Stabilization Time when setting up clocks at startup.<br><br>When FLL is enabled Software Standby and Deep Software Standby modes are not available. |
| Clock Settling Delay | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays for clocking scaling and transitions to ensure voltage supply stability. See the RA8T1 HWM (R01UH0996EJ0100) section 8.11.1 for details. |
| Sleep Mode Entry and Exit Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays before and after entering sleep modes to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK over 120MHz. See RA8T1 HWM (R01UH0996EJ0100) Section 10.7.10 for details. |
| RTOS Sleep on Idle | • Enabled<br>• Disabled | Disabled | Setting this option to Enabled will allow RTOS ports to enter CPU Sleep mode while idle. This should not be used when CPUCLK is configured over 120MHz. See RA8M1 HWM (R01UH0994EJ0100) Section 10.7.10 for details. |
| MSTP Change Delays | • Enabled<br>• Disabled | Enabled | Setting this option to Enabled will insert delays after setting a MSTP bit to ensure voltage supply stability. This is not required if you do not intend to run CPUCLK |

| | | | |
|---|---|---|---|
| | | | over 120MHz. See RA8T1 HWM (R01UH0996EJ0100) Section 10.4 for details. |
| Settling Delay (us) | Unit must be a non-negative integer | 150 | Specifies the length of the delay to be used for the Clock settling, Sleep mode, and MSTP change delays. |
| Cache settings | | | |
| Data cache | • Enabled<br>• Disabled | Disabled | Enable limited D-Cache support. See BSP usage notes for limitations. |
| I/O Ports | | | |
| I/O Ports > Voltage Mode | | | |
| VCC | • Low (VCC < 2.7 V)<br>• High (VCC >= 2.7 V) | High (VCC >= 2.7 V) | When VCC is lower than 2.7 V, set VCC I/O Ports Voltage Mode to Low. |
| VCC2 | • Low (VCC2 < 2.7 V)<br>• High (VCC2 >= 2.7 V) | High (VCC2 >= 2.7 V) | When VCC2 is lower than 2.7 V, set VCC2 I/O Ports Voltage Mode to Low. |
| Enable inline BSP IRQ functions | • Enabled<br>• Disabled | Enabled | Using static inline functions will slightly increase code size, but will slightly decrease cycles taken in ISRs in return. |
| Main Oscillator Wait Time | • 3 cycles<br>• 35 cycles<br>• 67 cycles<br>• 131 cycles<br>• 259 cycles<br>• 547 cycles<br>• 1059 cycles<br>• 2147 cycles<br>• 4291 cycles<br>• 8163 cycles | 8163 cycles | Number of cycles to wait for the main oscillator clock to stabilize. |

### Macros

| | |
|---|---|
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
|---|---|
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

#### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
| --- |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

#### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

#### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

## 5.1.5.39 RA8T2

BSP » MCU Board Support Package

### Detailed Description

### Macros

| | |
| --- | --- |
| #define | BSP_ELC_PERIPHERAL_MASK |

### Enumerations

| | |
| --- | --- |
| enum | elc_event_t |
| enum | elc_peripheral_t |

### Macro Definition Documentation

### ◆ BSP_ELC_PERIPHERAL_MASK

| #define BSP_ELC_PERIPHERAL_MASK |
| --- |
| Positions of event link set registers (ELSRs) available on this MCU |

### Enumeration Type Documentation

### ◆ elc_event_t

| enum elc_event_t |
| --- |
| Sources of event signals to be linked to other peripherals or the CPU<br><br>***Note***<br>*This list is device specific.* |

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
| --- |
| Possible peripherals to be linked to event signals<br><br>***Note***<br>*This list is device specific.* |

# 5.2 Modules

### Detailed Description

Modules are the smallest unit of software available in FSP. Each module implements one interface.

For more information on FSP Modules and Interfaces review FSP Modules, FSP Stacks and FSP Interfaces in the FSP Architecture section of this manual.

*Note*

*Not all modules are available for all MCUs. For more information, see the User's Manual for the specific MCU.*

### Organization of Module Sections

Each module within FSP has a detailed Users' Guide listed below. Each guide typically includes the following content:

- Functions: A list of all the API functions associated with the module
- Detailed Description: A short description of the module and the peripherals used
- Overview: An operational summary and a list of high level features provided by the module
- Configuration: A description of module specific settings available in the configuration tool

including clock and pin configurations
- Usage Notes: Module specific documentation and limitations
- Examples: Illustrative code snippets that help the user better understand API use and operation
- Data Structure and Enumeration: Definitions for data structures, enumerations and similar elements used by the module API
- Function Documentation: Details on each API function, including the function prototype, a function summary, a simple use example, list of return values and links to documentation for any needed parameter definitions

## Modules

| | |
|---|---|
| | Analog<br>Analog Modules. |
| | AI<br>Artificial Intelligence Modules. |
| | Audio<br>Audio Modules. |
| | Bootloader<br>Bootloader Modules. |
| | CapTouch<br>CapTouch Modules. |
| | Connectivity<br>Connectivity Modules. |
| | DSP<br>DSP Modules. |
| | Graphics<br>Graphics Modules. |
| | Input<br>Input Modules. |

| | |
|---|---|
| | Monitoring<br>Monitoring Modules. |
| | Motor<br>Motor Modules. |
| | Networking<br>Networking Modules. |
| | Power<br>Power Modules. |
| | RTOS<br>RTOS Modules. |
| | Security<br>Security Modules. |
| | Sensor<br>Sensor Modules. |
| | Storage<br>Storage Modules. |
| | System<br>System Modules. |
| | Timers<br>Timers Modules. |
| | Transfer<br>Transfer Modules. |

| | |
|---|---|
| | TrustZone<br>Arm TrustZone Modules. |

## 5.2.1 Analog

Modules

### Detailed Description

Analog Modules.

### Modules

| | |
|---|---|
| | ADC (r_adc)<br>Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface. |
| | ADC (r_adc_b)<br>Driver for the ADC_B peripheral on RA MCUs. This module implements the ADC Interface. |
| | ADC (r_adc_d)<br>Driver for ADC_D version of the ADC12 peripheral on RA MCUs. This module implements the ADC Interface. |
| | Comparator, High-Speed (r_acmphs)<br>Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface. |
| | Comparator, High-Speed (r_acmphs_b)<br>Driver for the ACMPHS_B peripheral on RA MCUs. This module implements the Comparator Interface. |
| | Comparator, Low-Power (r_acmplp)<br>Driver for the ACMPLP peripheral on RA MCUs. This module implements the Comparator Interface. |
| | DAC (r_dac) |

| | |
|---|---|
| | Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface. |
| | DAC8 (r_dac8)<br><br>Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface. |
| | DAC_B (r_dac_b)<br><br>Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface. |
| | Delta-Sigma Modulator Interface (r_dsmif)<br><br>Driver for the DSMIF peripheral on RA MCUs. This module implements the ADC Interface. |
| | Operational Amplifier (r_opamp)<br><br>Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface. |
| | SDADC Channel Configuration (r_sdadc)<br><br>Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface. |
| | SDADC_B Channel Configuration (r_sdadc_b)<br><br>Driver for the SDADC_B peripheral on RA MCUs. This module implements the ADC Interface. |

#### 5.2.1.1 ADC (r_adc)

Modules » Analog

**Functions**

| | |
|---|---|
| fsp_err_t | R_ADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_channel_cfg) |
| fsp_err_t | R_ADC_CallbackSet (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, |

| | |
|---|---|
| | adc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_ADC_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id) |
| fsp_err_t | R_ADC_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_ADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_ADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_ADC_SampleStateCountSet (adc_ctrl_t *p_ctrl, adc_sample_state_t *p_sample) |
| fsp_err_t | R_ADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_ADC_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend) |
| fsp_err_t | R_ADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset) |

### Detailed Description

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The ADC module supports the following features:

- 12, 14, or 16 bit maximum resolution depending on the MCU
- Configure scans to include:
  - Multiple analog channels
  - Temperature sensor channel
  - Voltage sensor channel
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
  - External scan triggers from the ADTRGn port pins
- Configurable scan mode:

    - Single scan mode, where each trigger starts a single scan
    - Continuous scan mode, where all channels are scanned continuously
    - Group scan mode, where channels are grouped into group A and group B. The groups can be assigned different start triggers, and group A can be given priority over group B. When group A has priority over group B, a group A trigger suspends an ongoing group B scan.
- Supports adding and averaging converted samples
- Optional callback when scan completes
- Sample and hold support
- Double-trigger support
- Hardware comparator with interrupt and event output

# Configuration

### Build Time Configurations for r_adc

The following build time configurations are defined in fsp_cfg/r_adc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > ADC (r_adc)

This module can be added to the Stacks tab via New Stack > Analog > ADC (r_adc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_adc0 | Module name |
| Unit | Unit must be a non-negative integer | 0 | Specifies the ADC Unit to be used. |
| Resolution | MCU Specific Options | | Specifies the conversion resolution for this unit. |
| Alignment | MCU Specific Options | | Specifies the conversion result alignment. |
| Clear after read | • Off<br>• On | On | Specifies if the result register will be automatically cleared after the conversion result is read. |
| Mode | • Single Scan | Single Scan | Specifies the mode that |

| | | | |
|---|---|---|---|
| | • Continuous Scan<br>• Group Scan | | this ADC unit is used in. |
| Double-trigger | • Disabled<br>• Enabled<br>• Enabled (extended mode) | Disabled | When enabled, the scan-end interrupt for Group A is only thrown on every second scan. Extended double-trigger mode (single-scan only) triggers on both ELC events, allowing (for example) a scan on two different timer compare match values.<br><br>In group mode Group B is unaffected. |
| Input | | | |
| Input > Sample and Hold | | | |
| Sample and Hold Channels (Available only on selected MCUs) | • Channel 0<br>• Channel 1<br>• Channel 2 | | Specifies if this channel is included in the Sample and Hold Mask. |
| Sample Hold States (Applies only to channels 0, 1, 2) | Must be a valid non-negative integer with configurable value 4 to 255 | 24 | Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit |
| Input > Window Compare | | | |
| Input > Window Compare > Window A | | | |
| Enable | • Disabled<br>• Enabled | Disabled | Enable or disable comparison with Window A. |
| Channels to compare (channel availability varies by MCU and unit) | Refer to the RA Configuration tool for available options. | | Select channels to be compared to Window A. |
| Channel comparison mode (channel availability varies by MCU and unit) | Refer to the RA Configuration tool for available options. | | Checking a box sets the comparison mode for that channel to Greater Than or Inside Window depending on whether Window Mode is disabled or enabled (respectively). If left unchecked the comparison mode will |

| | | | |
|---|---|---|---|
| | | | likewise be Less Than or Outside Window (respectively). |
| Lower Reference | Must be a positive 16-bit integer. | 0 | Set the lower comparison value. |
| Upper Reference | Must be a positive 16-bit integer. | 0 | Set the upper comparison value. |
| Input > Window Compare > Window B | | | |
| Enable | • Disabled<br>• Enabled | Disabled | Enable or disable comparison with Window B. |
| Channel to compare (channel availability varies by MCU and unit) | Refer to the RA Configuration tool for available options. | Channel 0 | Select a channel to be compared to Window B. |
| Comparison mode | • Less Than or Outside Window<br>• Greater Than or Inside Window | Less Than or Outside Window | Select the comparison mode for Window B. For each option, the first condition applies when Window Mode is disabled and the second option applies when Window Mode is enabled. |
| Lower Reference | Must be a positive 16-bit integer. | 0 | Set the lower comparison value. |
| Upper Reference | Must be a positive 16-bit integer. | 0 | Set the upper comparison value. |
| Window Mode | • Disabled<br>• Enabled | Disabled | When disabled, ADC values will be compared only with the lower reference on each comparator. When enabled, both the lower and upper reference values will be used to create a comparison window. |
| Event Output | • OR<br>• XOR<br>• AND | OR | Select how comparison results should be composited for event output. |
| Channel Scan Mask (channel availability varies by MCU) | Refer to the RA Configuration tool for available options. | | In Normal mode of operation, this bitmask field specifies the channels that are enabled in that ADC unit. In group mode, |

| | | | |
|---|---|---|---|
| | | | this field specifies which channels belong to group A. |
| Group B Scan Mask (channel availability varies by MCU) | Refer to the RA Configuration tool for available options. | | In group mode, this field specifies which channels belong to group B. |
| Add/Average Count | MCU Specific Options | | Specifies if addition or averaging needs to be done for any of the channels in this unit. |
| Reference Voltage control | MCU Specific Options | | Specify VREFH/VREFADC output voltage control. |
| Addition/Averaging Mask (channel availability varies by MCU and unit) | Refer to the RA Configuration tool for available options. | | Select channels to include in the Addition/Averaging Mask |
| Interrupts | | | |
| Normal/Group A Trigger | MCU Specific Options | | Specifies the trigger type to be used for this unit. Triggers that specify ADC Unit must be selected for correct ADC unit to operate correctly. |
| Group B Trigger | MCU Specific Options | | Specifies the trigger for Group B scanning in group scanning mode. This event is also used to trigger Group A in extended double-trigger mode. Triggers that specify ADC Unit must be selected for correct ADC unit to operate correctly. |
| Group Priority (Valid only in Group Scan Mode) | • Group A cannot interrupt Group B<br>• Group A can interrupt Group B; Group B scan restarts at next trigger<br>• Group A can interrupt Group B; Group B scan restarts immediately | Group A cannot interrupt Group B | Determines whether an ongoing group B scan can be interrupted by a group A trigger, whether it should abort on a group A trigger, or if it should pause to allow group A scan and restart immediately after group A scan is complete. |

| | | | |
|---|---|---|---|
| | • Group A can interrupt Group B; Group B scan restarts immediately and scans continuously | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes. |
| Scan End Interrupt Priority | MCU Specific Options | | Select scan end interrupt priority. |
| Scan End Group B Interrupt Priority | MCU Specific Options | | Select group B scan end interrupt priority. |
| Window Compare A Interrupt Priority | MCU Specific Options | | Select Window Compare A interrupt priority. |
| Window Compare B Interrupt Priority | MCU Specific Options | | Select Window Compare B interrupt priority. |
| Extra | | | |
| ADC Ring Buffer | MCU Specific Options | | ADC Ring Buffer to be used only with DMAC transfers, keep this property disabled for normal ADC operations. When enabled, ADC converted data is stored in ADBUF registers in place of ADDR registers. The read API will not read from this location for normal ADC operations. |

### Clock Configuration

The ADC clock is PCLKC if the MCU has PCLKC, or PCLKD otherwise.

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA0E1 | ICLK/PCLKB |

| | |
|---|---|
| RA0E2 | ICLK/PCLKB |
| RA0L1 | ICLK/PCLKB |
| RA2A1 | PCLKD |
| RA2A2 | PCLKD |
| RA2E1 | PCLKD |
| RA2E2 | PCLKD |
| RA2E3 | PCLKD |
| RA2L1 | PCLKD |
| RA2L2 | PCLKD |
| RA2T1 | PCLKD |
| RA4C1 | PCLKC |
| RA4E1 | PCLKC |
| RA4E2 | PCLKC |
| RA4L1 | PCLKC |
| RA4M1 | PCLKC |
| RA4M2 | PCLKC |
| RA4M3 | PCLKC |
| RA4T1 | PCLKC |
| RA4W1 | PCLKC |
| RA6E1 | PCLKC |
| RA6E2 | PCLKC |
| RA6M1 | PCLKC |
| RA6M2 | PCLKC |
| RA6M3 | PCLKC |
| RA6M4 | PCLKC |
| RA6M5 | PCLKC |
| RA6T1 | PCLKC |
| RA6T3 | PCLKC |
| RA8D1 | PCLKC |
| RA8E1 | PCLKC |
| RA8E2 | PCLKC |
| RA8M1 | PCLKC |
| RA8T1 | PCLKC |

The ADC clock must be at least 1 MHz when the ADC is used. Many MCUs also have PCLK ratio restrictions when the ADC is used. For details on PCLK ratio restrictions, reference the footnotes in the second table of the Clock Generation Circuit chapter of the MCU User's Manual (for example, Table "Specifications of the clock generation circuit for the internal clocks" in the CGC section of the relevant hardware manual).

### Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC.

ADTRG0 and ADTRG1 can be used to start scans with an external trigger for unit 0 and 1 respectively. When external triggers are used, ADC scans begin on the falling edge of the ADTRG pin.

## Usage Notes

### Sample Hold

Enabling the sample and hold functionality reduces the maximum scan frequency because the sample and hold time is added to each scan. Refer to the hardware manual for details on the sample and hold time.

### ADC Operational Modes

The driver supports three operation modes: single-scan, continuous-scan, and group-scan modes. In each mode, analog channels are converted in ascending order of channel number, followed by scans of the temperature sensor and voltage sensor if they are included in the mask of channels to scan.

### Single-scan Mode

In single scan mode, one or more specified channels are scanned once per trigger.

### Continuous-scan Mode

In continuous scan mode, a single trigger is required to start the scan. Scans continue until R_ADC_ScanStop() is called.

*Note*

*1) To help ensure a responsive system, developers should consider system clock speed, ADCLK speed, and callback processing time. In particular, using a scan-end callback with a high scan rate relative to core clocks (for example, in continuous scan mode) may result in constant or high-frequency interrupts and is not recommended.*
*2) On some MCUs, scanning the temperature sensor or internal reference voltage in continuous or group scan modes is prohibited. Check the "ADC12 specifications" section of the device's User Manual to see if this restriction applies.*

### Group-scan Mode

Group-scan mode allows the application to allocate channels to one of two groups (A and B). Conversion begins when the specified ELC start trigger for that group is received.

With the priority configuration parameter, you can optionally give group A priority over group B. If group A has priority over group B, a group B scan is interrupted when a group A scan trigger occurs. The following options exist for group B when group A has priority:

- To restart the interrupted group B scan after the group A scan completes.
- To wait for another group B trigger and forget the interrupted scan.
- To continuously scan group B and suspend scanning group B only when a group A trigger is received.

  *Note*

  *If this option is selected, group B scanning begins immediately after R_ADC_ScanCfg(). Group A scan triggers must be enabled by R_ADC_ScanStart() and can be disabled by R_ADC_ScanStop(). Group B scans can only be disabled by reconfiguring the group A priority to a different mode.*

### Double-triggering

When double-triggering is enabled a single channel is selected to be scanned twice before an interrupt is thrown. The first scan result when using double-triggering is always saved to the selected channel's data register. The second result is saved to the data duplexing register (ADC_CHANNEL_DUPLEX).

Double-triggering uses Group A; only one channel can be selected when enabled. No other scanning is possible on Group A while double-trigger mode is selected. In addition, any special ADC channels (such as temperature sensors or voltage references) are not valid double-trigger channels.

When extended double-triggering is enabled, both ADC input (ELC) events are routed to Group A. The interrupt is still thrown after every two scans regardless of the triggering event(s). While the first and second scan are saved to the selected ADC data register and the ADC duplexing register as before, scans associated with event A and B are additionally copied into duplexing register A and B, respectively (ADC_CHANNEL_DUPLEX_A and ADC_CHANNEL_DUPLEX_B).

### When Interrupts Are Not Enabled

If interrupts are not enabled, the R_ADC_StatusGet API can be used to poll the ADC to determine when the scan has completed. The read API function is used to access the converted ADC result. This applies to both normal scans and calibration scans for MCUs that support calibration.

### Window Compare Function

The ADC contains comparators that allow scan data to be compared to user-provided reference values. When a value meets the configured condition an interrupt and/or an ELC event can be produced.

Each unit has two configurable comparison units, Window A and Window B. Window A allows for configuring multiple simultaneous channels to compare while Window B only allows one channel at a time.

The window compare function can be configured both through the RA Configuration tool and at runtime by providing a pointer to an adc_window_cfg_t struct to adc_channel_cfg_t::p_window_cfg when calling R_ADC_ScanCfg. The available comparison modes are shown below:

| Window setting | Channel mode 0 | Channel mode 1 |
|---|---|---|
| Disabled | Scan < Low Ref | Scan > Low Ref |
| Enabled | (Scan < Low Ref) OR (Scan > High Ref) | Low Ref < Scan < High Ref |

*Note*

*The window setting applies to all channels configured on a unit.*

### Event Output Function

To use the event output of compare function, Set the mode of operation to Single Scan Mode. The compare function can be configured by enabling both Window A and Window B. Select the channel of Window A. Select Window B channel to NOT SELECTED. Set the upper and lower reference values for both Window A and Window B.

*Note*

*When using ADC12 in Snooze Mode, both Compare Window A and Compare Window B must be enabled.*

### Sample-State Count Setting

The application program can modify the setting of the sample-state count for analog channels by calling the R_ADC_SampleStateCountSet() API function. The application program only needs to modify the sample-state count settings from their default values to increase the sampling time. This can be either because the impedance of the input signal is too high to secure sufficient sampling time under the default setting or if the ADCLK is too slow. To modify the sample-state count for a given channel, set the channel number and the number of states when calling the R_ADC_SampleStateCountSet() API function. Valid sample state counts are 7-255.

*Note*

*Although the hardware supports a minimum number of sample states of 5, some MCUs require 7 states, so the minimum is set to 7. At the lowest supported ADC conversion clock rate (1 MHz), these extra states will lead to, at worst case, a 2 microsecond increase in conversion time. At 60 MHz the extra states will add 33.4 ns to the conversion time.*

If the sample state count needs to be changed for multiple channels, the application program must call the R_ADC_SampleStateCountSet() API function repeatedly, with appropriately modified arguments for each channel.

If the ADCLK frequency changes, the sample states may need to be updated.

### Sample States for Temperature Sensor and Internal Voltage Reference

Sample states for the temperature sensor and the internal reference voltage are calculated during R_ADC_ScanCfg() based on the ADCLK frequency at the time. The sample states for the temperature sensor and internal voltage reference cannot be updated with R_ADC_SampleStateCountSet(). If the ADCLK frequency changes, call R_ADC_ScanCfg() before using the temperature sensor or internal reference voltage again to ensure the sampling time for the temperature sensor and internal voltage reference is optimal.

### Selecting Reference Voltage

The ADC high-potential and low-potential reference voltages may be configured for selected MCU's. Please refer to the RA Configuration editor in e² studio for further details.

*Note*

*When using VREFADC, a stabilization time of 1500us is required after call for R_ADC_Open(). Consult section "Selecting Reference Voltage" in the ADC section of the relevant hardware manual for details.*
*When the internal reference voltage is selected as the high-potential reference voltage, the Low-power A/D Conversion mode will automatically be selected. Consult Section "Selecting Reference Voltage" in the ADC section of the relevant hardware manual for details.*

### Using the Temperature Sensor with the ADC

The ADC HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula, T = (Vs - V1)/slope + T1, where:

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- T2: Temperature at the experimental measurement of another point (degrees C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/degrees C); slope = (V2 - V1)/(T2 - T1)

*Note*

*The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.*

### Reading CTSU TSCAP with ADC

Some MCUs support reading CTSU TSCAP with ADC. CTSU TSCAP is connected to ADC0 channel 16. Use existing enums for channel 16 to set sample states for the sensor connected to CTSU TSCAP, enable scanning of CTSU TSCAP, and read results for CTSU TSCAP.

### Usage Notes for ADC16

#### Calibration

Calibration is required to use the ADC16 peripheral. When using this driver on an MCU that has ADC16, call R_ADC_Calibrate() after open, and prior to any other function.

### Range of ADC16 Results

The range of the ADC16 is from 0 (lowest) to 0x7FFF (highest) when used in single-ended mode. This driver only supports single ended mode.

### Limitations

Developers should be aware of the following limitations when using the ADC:

- If the ratio between the core clock and the ADC source clock is higher than 32:1, call R_ADC_ScanStop before calling R_ADC_Close if ADC is still running.
- When using the Window Compare function:
  - Only Single Scan mode may be configured when match or mismatch ELC events are used.
  - When one compare window is configured to check the temperature sensor or voltage reference the other window cannot be used.
  - Both windows cannot reference the same channel.
  - When using ADC in Snooze Mode, both Window A and Window B must be enabled.

# Examples

### Basic Example

This is a basic example of minimal use of the ADC in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_ScanCfg. */
const adc_channel_cfg_t g_adc0_channel_cfg =
{
    .scan_mask          = ADC_MASK_CHANNEL_0 | ADC_MASK_CHANNEL_1,
    .scan_mask_group_b = 0,
    .priority_group_a   = (adc_group_a_t) 0,
    .add_mask          = 0,
    .sample_hold_mask   = 0,
    .sample_hold_states = 0,
};
void adc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
```

```
    }
 /* Read converted data. */
    uint16_t channel1_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
    assert(FSP_SUCCESS == err);
}
```

### Temperature Sensor Example

This example shows how to calculate the MCU temperature using the ADC and the temperature sensor.

```
#define ADC_EXAMPLE_CALIBRATION_DATA_RA6M1 (0x7D5)
#define ADC_EXAMPLE_VCC_MICROVOLT (3300000)
#define ADC_EXAMPLE_TEMPERATURE_RESOLUTION (12U)
#define ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE (127)
void adc_temperature_example (void)
{
 /* The following example calculates the temperature using the data provided in the
section
  * See "Preparation for Using the Temperature Sensor" in the TSN secton of the
relevant hardware manual. */
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable temperature sensor. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
```

```
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t temperature_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_TEMPERATURE,
&temperature_conversion_result);
    assert(FSP_SUCCESS == err);
 /* If the MCU does not provide calibration data, use the value in the hardware
manual or determine it
  * experimentally. */
 /* Get Calibration data from the MCU if available. */
    int32_t    reference_calibration_data;
 adc_info_t adc_info;
    (void) R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
    reference_calibration_data = (int32_t) adc_info.calibration_data;
 /* NOTE: The slope of the temperature sensor varies from sensor to sensor. Renesas
recommends calculating
  * the slope of the temperature sensor experimentally.
  *
  * This example uses the typical slope provided in Table "TSN characteristics"
  * in the relevant hardware manual. */
    int32_t slope_uv_per_c = BSP_FEATURE_TSN_SLOPE;
 /* Formula for calculating temperature copied from section "Preparation for Using
the Temperature Sensor"
  * of the relevant hardware manual:
  *
  * In this MCU, the TSCDR register stores the temperature value (CAL127) of the
temperature sensor measured
  * under the condition Ta = Tj = 127 C and AVCC0 = 3.3 V. By using this value as the
sample measurement result
```

```
  * at the first point, preparation before using the temperature sensor can be
omitted.
  *
  * If V1 is calculated from CAL127,
  * V1 = 3.3 * CAL127 / 4096 [V]
  *
  * Using this, the measured temperature can be calculated according to the following
formula.
  *
  * T = (Vs - V1) / Slope + 127 [C]
  * T: Measured temperature (C)
  * Vs: Voltage output by the temperature sensor when the temperature is measured (V)
  * V1: Voltage output by the temperature sensor when Ta = Tj = 127 C and AVCC0 = 3.3
V (V)
  * Slope: Temperature slope given in Table 52.38 / 1000 (V/C)
  */
    int32_t v1_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                  reference_calibration_data;
    int32_t vs_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                  temperature_conversion_result;
    int32_t temperature_c = (vs_uv - v1_uv) / slope_uv_per_c +
ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE;
 /* Expect room temperature, break if temperature is outside the range of 20 C to 25
C. */
 if ((temperature_c < 20) || (temperature_c > 25))
    {
       __BKPT(0);
    }
}
```

### Double-Trigger Example

This example demonstrates reading data from a double-trigger scan. A flag is used to wait for a

callback event. Two scans must occur before the callback is called. These results are read via R_ADC_Read using the selected channel enum value as well as ADC_CHANNEL_DUPLEX.

```
volatile bool scan_complete_flag = false;
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    scan_complete_flag = true;
}
void adc_double_trigger_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable double-trigger channel. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable scan triggering from ELC events. */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. Two scans must be triggered before a callback
occurs. */
    scan_complete_flag = false;
 while (!scan_complete_flag)
    {
 /* Wait for callback to set flag. */
    }
 /* Read converted data from both scans. */
    uint16_t channel1_conversion_result_0;
    uint16_t channel1_conversion_result_1;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result_0);
    assert(FSP_SUCCESS == err);
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_DUPLEX,
&channel1_conversion_result_1);
```

```
    assert(FSP_SUCCESS == err);
}
```

### ADC-DMAC Repeat-Block Transfer Example

This example demonstrates writing multiple data from DAC peripheral to ADC channels and storing the data in memory through DMAC using Repeat-Block Transfer mode. It creates single block to multiple ring buffer type of transfer topology. Ping-Pong mechanism is used to read the data from memory in between the transfers. This example is valid only for MCUs that have ADBUF.

```
 #define ADC_DMAC_EXAMPLE_DATA_LOW (0U)
 #define ADC_DMAC_EXAMPLE_DATA_HIGH (0x000FU)
 #define ADC_DMAC_EXAMPLE_DELAY_1000_MS (1000U)
 #define ADC_DMAC_EXAMPLE_NUM_PING_PONG_BUFFERS (2)
static uint16_t g_adc_dmac_example_buffer[ADC_DMAC_EXAMPLE_NUM_PING_PONG_BUFFERS][
    ADC_DMAC_EXAMPLE_ADC_CHANNELS_PER_BLOCK][ADC_DMAC_EXAMPLE_SAMPLES_PER_CHANNEL];
// Destination buffer for DMAC transfers
static volatile uint16_t g_adc_dmac_example_ping_pong_index = 0U;
static volatile void * gp_read_data;
/* DMAC callback */
void adc_dmac_callback (dmac_callback_args_t * p_args)
{
    (void) p_args;
 /* Store the pointer to the last buffer that was written
  * An array of data for the first enabled channel is at
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0],
  * an array of data for the next channel is at
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][1][0], etc.
  */
    gp_read_data =
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0];
 /* Select the other ping-pong buffer which is free for writing */
    g_adc_dmac_example_ping_pong_index = !g_adc_dmac_example_ping_pong_index;
 /* Reset the destination pointer and DMAC peripheral */
 R_DMAC_Reset(&g_transfer0_ctrl,
               NULL,
```

```
                (void *)
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0],
                ADC_DMAC_EXAMPLE_SAMPLES_PER_CHANNEL);
 FSP_PARAMETER_NOT_USED(gp_read_data);
}
void adc_dmac_repeat_block_transfer_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open ADC Module and configure the channels */
 /* Enable the ADBUF property from configurations */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Open DMAC channel for repeat-block transfer with following configurations
  * (1) Destination address as
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0]
  * (2) Enable end of transfer interrupt
  * (3) Configure source address mode as incremented and destination address mode as
offset addition,
  * fixed to address of ADBUF register by configurator with ADC-DMAC module
  * (4) Configure source buffer size as total size of source buffer - See "DMSBS :
DMA Source Buffer Size Register"
  * description in the DMAC section of the relevant hardware manual for source buffer
size limitations
  * (5) Configure transfer mode as Repeat-Block mode
  * (6) See "DMDBS : DMA Destination Buffer Size Register" description in the DMAC
section of the relevant hardware
  * manual for total number of blocks which decides destination buffer size
  * (7) Number of blocks is determined by the samples per channel property for ADC-
DMAC module
  * (8) Size of block is determined using the enabled ADC channels in the
configurator when using ADC-DMAC-module
  * (9) Configure DMAC activation source as A/D scan end interrupt
```

```
 */
   err = R_DMAC_Open(&g_transfer0_ctrl, &g_transfer0_cfg);
   assert(FSP_SUCCESS == err);
   err = R_DMAC_Enable(&g_transfer0_ctrl);
   assert(FSP_SUCCESS == err);
   uint16_t    count = ADC_DMAC_EXAMPLE_DATA_LOW;
 adc_status_t adc_status;
 /* Trigger the ADC scan for "count" times, this can be replaced by triggering the
ADC using a timer */
 while (count <= (uint16_t) ADC_DMAC_EXAMPLE_DATA_HIGH)
   {
 /* Scan the data with ADC channels*/
      err = R_ADC_ScanStart(&g_adc0_ctrl);
     assert(FSP_SUCCESS == err);
 /* Wait for conversion to complete */
      uint16_t timeout = UINT16_MAX;
      adc_status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while ((ADC_STATE_SCAN_IN_PROGRESS == adc_status.state) && (timeout > 0U))
     {
           timeout--;
 R_ADC_StatusGet(&g_adc0_ctrl, &adc_status);
     }
 R_BSP_SoftwareDelay(ADC_DMAC_EXAMPLE_DELAY_1000_MS, BSP_DELAY_UNITS_MICROSECONDS);
      count++;
   }
}
```

### Window Compare Example

This example shows how to configure the window compare function at runtime as well as how to handle events and obtain comparison results through a callback.

```
adc_window_cfg_t g_adc0_window_cfg =
{
 /* Enable Window A and Window B; enable Window mode */
```

```
    .compare_cfg       =
       (adc_compare_cfg_t) (ADC_COMPARE_CFG_A_ENABLE | ADC_COMPARE_CFG_B_ENABLE |
ADC_COMPARE_CFG_WINDOW_ENABLE),
 /* Compare scan values from Channels 0 and 1 */
    .compare_mask       = ADC_MASK_CHANNEL_0 | ADC_MASK_CHANNEL_1,
 /* Set Channel 1 condition to be inside the window instead of outside */
    .compare_mode_mask = ADC_MASK_CHANNEL_1,
 /* Set reference voltage levels for Window A */
    .compare_ref_low = ADC_SCAN_MAX / 3,
    .compare_ref_high = ADC_SCAN_MAX * 2 / 3,
 /* Configure Window B to compare Channel 2 (inside window) */
    .compare_b_channel = ADC_WINDOW_B_CHANNEL_2,
    .compare_b_mode    = ADC_WINDOW_B_MODE_GREATER_THAN_OR_INSIDE,
 /* Set reference voltage levels for Window B */
    .compare_b_ref_low = ADC_SCAN_MAX / 4,
    .compare_b_ref_high = ADC_SCAN_MAX * 3 / 4,
};
void adc0_callback (adc_callback_args_t * p_args)
{
 if (ADC_EVENT_WINDOW_COMPARE_A == p_args->event)
    {
 /* Get channel that met the comparison criteria */
       adc_channel_t channel = p_args->channel;
 /* Process event here */
 FSP_PARAMETER_NOT_USED(channel);
    }
 else if (ADC_EVENT_WINDOW_COMPARE_B == p_args->event)
    {
 /* Process Window B events here */
    }
 else
    {
 /* ... */
    }
```

```
}
void adc_window_compare_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the ADC module */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    assert(FSP_SUCCESS == err);
 /* Set the window compare configuration in the channel config */
    g_adc0_channel_runtime_cfg.p_window_cfg = &g_adc0_window_cfg;
 /* The window compare function is configured as part of the scan configuration */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_runtime_cfg);
    assert(FSP_SUCCESS == err);
 /* Main program loop - scan the ADC every second */
 while (1)
    {
 /* Start a scan */
       err = R_ADC_ScanStart(&g_adc0_ctrl);
      assert(FSP_SUCCESS == err);
 /* Delay; any compare events will be handled by the callback */
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | adc_sample_state_t |
| struct | adc_window_cfg_t |
| struct | adc_extended_cfg_t |
| struct | adc_channel_cfg_t |
| struct | adc_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | adc_mask_t |
| enum | adc_add_t |

| | |
|---|---|
| enum | adc_clear_t |
| enum | adc_vref_control_t |
| enum | adc_sample_state_reg_t |
| enum | adc_compare_cfg_t |
| enum | adc_window_b_channel_t |
| enum | adc_window_b_mode_t |
| enum | adc_group_a_t |
| enum | adc_double_trigger_t |
| enum | adc_start_source_t |

## Data Structure Documentation

### ◆ adc_sample_state_t

| struct adc_sample_state_t | | |
|---|---|---|
| ADC sample state configuration | | |
| Data Fields | | |
| adc_sample_state_reg_t | reg_id | Sample state register ID. |
| uint8_t | num_states | Number of sampling states for conversion. Ch16-20/21 use the same value. |

### ◆ adc_window_cfg_t

| struct adc_window_cfg_t | | |
|---|---|---|
| ADC Window Compare configuration | | |
| Data Fields | | |
| uint32_t | compare_mask | Channel mask to compare with Window A. |
| uint32_t | compare_mode_mask | Per-channel condition mask for Window A. |
| adc_compare_cfg_t | compare_cfg | Window Compare configuration. |
| uint16_t | compare_ref_low | Window A lower reference value. |
| uint16_t | compare_ref_high | Window A upper reference value. |
| uint16_t | compare_b_ref_low | Window B lower reference |

| | | |
|---|---|---|
| | | value. |
| uint16_t | compare_b_ref_high | Window A upper reference value. |
| adc_window_b_channel_t | compare_b_channel | Window B channel. |
| adc_window_b_mode_t | compare_b_mode | Window B condition setting. |

### ◆ adc_extended_cfg_t

| struct adc_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for ADC. | | |
| Data Fields | | |
| adc_add_t | add_average_count | Add or average samples. |
| adc_clear_t | clearing | Clear after read. |
| adc_start_source_t | trigger | Trigger source for ADC. |
| adc_start_source_t | trigger_group_b | Trigger source for ADC group B; valid only for group mode. |
| adc_double_trigger_t | double_trigger_mode | Double-trigger mode setting. |
| adc_vref_control_t | adc_vref_control | VREFADC output voltage control. |
| uint8_t | enable_adbuf | Enable ADC Ring Buffer, Valid only to use along with DMAC transfer. |
| IRQn_Type | window_a_irq | IRQ number for Window Compare A interrupts. |
| IRQn_Type | window_b_irq | IRQ number for Window Compare B interrupts. |
| uint8_t | window_a_ipl | Priority for Window Compare A interrupts. |
| uint8_t | window_b_ipl | Priority for Window Compare B interrupts. |

### ◆ adc_channel_cfg_t

| struct adc_channel_cfg_t | | |
|---|---|---|
| ADC channel(s) configuration | | |
| Data Fields | | |
| uint32_t | scan_mask | Channels/bits: bit 0 is ch0; bit 15 is ch15. |
| uint32_t | scan_mask_group_b | Valid for group modes. |
| uint32_t | add_mask | Valid if add enabled in Open(). |
| adc_window_cfg_t * | p_window_cfg | Pointer to Window Compare configuration. |

| | | |
|---|---|---|
| adc_group_a_t | priority_group_a | Valid for group modes. |
| uint8_t | sample_hold_mask | Channels/bits 0-2. |
| uint8_t | sample_hold_states | Number of states to be used for sample and hold. Affects channels 0-2. |

### ◆ adc_instance_ctrl_t

| struct adc_instance_ctrl_t |
|---|
| ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). |

## Enumeration Type Documentation

### ◆ adc_mask_t

| enum adc_mask_t | |
|---|---|
| For ADC Scan configuration adc_channel_cfg_t::scan_mask, adc_channel_cfg_t::scan_mask_group_b, adc_channel_cfg_t::add_mask and adc_channel_cfg_t::sample_hold_mask. Use bitwise OR to combine these masks for desired channels and sensors. | |
| Enumerator | |
| ADC_MASK_OFF | No channels selected. |
| ADC_MASK_CHANNEL_0 | Channel 0 mask. |
| ADC_MASK_CHANNEL_1 | Channel 1 mask. |
| ADC_MASK_CHANNEL_2 | Channel 2 mask. |
| ADC_MASK_CHANNEL_3 | Channel 3 mask. |
| ADC_MASK_CHANNEL_4 | Channel 4 mask. |
| ADC_MASK_CHANNEL_5 | Channel 5 mask. |
| ADC_MASK_CHANNEL_6 | Channel 6 mask. |
| ADC_MASK_CHANNEL_7 | Channel 7 mask. |
| ADC_MASK_CHANNEL_8 | Channel 8 mask. |
| ADC_MASK_CHANNEL_9 | Channel 9 mask. |
| ADC_MASK_CHANNEL_10 | Channel 10 mask. |
| ADC_MASK_CHANNEL_11 | Channel 11 mask. |

| | |
|---|---|
| ADC_MASK_CHANNEL_12 | Channel 12 mask. |
| ADC_MASK_CHANNEL_13 | Channel 13 mask. |
| ADC_MASK_CHANNEL_14 | Channel 14 mask. |
| ADC_MASK_CHANNEL_15 | Channel 15 mask. |
| ADC_MASK_CHANNEL_16 | Channel 16 mask. |
| ADC_MASK_CHANNEL_17 | Channel 17 mask. |
| ADC_MASK_CHANNEL_18 | Channel 18 mask. |
| ADC_MASK_CHANNEL_19 | Channel 19 mask. |
| ADC_MASK_CHANNEL_20 | Channel 20 mask. |
| ADC_MASK_CHANNEL_21 | Channel 21 mask. |
| ADC_MASK_CHANNEL_22 | Channel 22 mask. |
| ADC_MASK_CHANNEL_23 | Channel 23 mask. |
| ADC_MASK_CHANNEL_24 | Channel 24 mask. |
| ADC_MASK_CHANNEL_25 | Channel 25 mask. |
| ADC_MASK_CHANNEL_26 | Channel 26 mask. |
| ADC_MASK_CHANNEL_27 | Channel 27 mask. |
| ADC_MASK_CHANNEL_28 | Channel 28 mask. |
| ADC_MASK_TEMPERATURE | Temperature sensor channel mask. |
| ADC_MASK_VOLT | Voltage reference channel mask. |
| ADC_MASK_SENSORS | All sensor channel mask. |

### ◆ adc_add_t

| enum adc_add_t | |
|---|---|
| ADC data sample addition and averaging options | |
| Enumerator | |
| ADC_ADD_OFF | Addition turned off for channels/sensors. |
| ADC_ADD_TWO | Add two samples. |
| ADC_ADD_THREE | Add three samples. |
| ADC_ADD_FOUR | Add four samples. |
| ADC_ADD_SIXTEEN | Add sixteen samples. |
| ADC_ADD_AVERAGE_TWO | Average two samples. |
| ADC_ADD_AVERAGE_FOUR | Average four samples. |
| ADC_ADD_AVERAGE_EIGHT | Average eight samples. |
| ADC_ADD_AVERAGE_SIXTEEN | Add sixteen samples. |

### ◆ adc_clear_t

| enum adc_clear_t | |
|---|---|
| ADC clear after read definitions | |
| Enumerator | |
| ADC_CLEAR_AFTER_READ_OFF | Clear after read off. |
| ADC_CLEAR_AFTER_READ_ON | Clear after read on. |

### ◆ adc_vref_control_t

| enum adc_vref_control_t | |
|---|---|
| Enumerator | |
| ADC_VREF_CONTROL_VREFH | VREFAMPCNT reset value. VREFADC Output voltage is Hi-Z. |
| ADC_VREF_CONTROL_1_5V_OUTPUT | BGR turn ON. VREFADC Output voltage is 1.5 V. |
| ADC_VREF_CONTROL_2_0V_OUTPUT | BGR turn ON. VREFADC Output voltage is 2.0 V. |
| ADC_VREF_CONTROL_2_5V_OUTPUT | BGR turn ON. VREFADC Output voltage is 2.5 V. |
| ADC_VREF_CONTROL_AVCC0_AVSS0 | High potential is AVCC0, low potential is AVSS0. |
| ADC_VREF_CONTROL_VREFH0_AVSS0 | High potential is VREFH0, low potential is AVSS0. |
| ADC_VREF_CONTROL_IVREF_AVSS0 | High potential is internal reference voltage, low potential is AVSS0. When the high potential is set to the internal reference voltage, wait 5 us after R_ADC_Open() to start an ADC measurement. |
| ADC_VREF_CONTROL_AVCC0_VREFL0 | High potential is AVCC0, low potential is VREFL0. |
| ADC_VREF_CONTROL_VREFH0_VREFL0 | High potential is VREFH0, low potential is VREFL0. |
| ADC_VREF_CONTROL_IVREF_VREFL0 | High potential is internal reference voltage, low potential is VREFL0. When the high potential is set to the internal reference voltage, wait 5 us after R_ADC_Open() to start an ADC measurement. |

### ◆ adc_sample_state_reg_t

| enum adc_sample_state_reg_t | |
|---|---|
| ADC sample state registers | |
| Enumerator | |
| ADC_SAMPLE_STATE_CHANNEL_0 | Sample state register channel 0. |
| ADC_SAMPLE_STATE_CHANNEL_1 | Sample state register channel 1. |
| ADC_SAMPLE_STATE_CHANNEL_2 | Sample state register channel 2. |
| ADC_SAMPLE_STATE_CHANNEL_3 | Sample state register channel 3. |
| ADC_SAMPLE_STATE_CHANNEL_4 | Sample state register channel 4. |
| ADC_SAMPLE_STATE_CHANNEL_5 | Sample state register channel 5. |
| ADC_SAMPLE_STATE_CHANNEL_6 | Sample state register channel 6. |
| ADC_SAMPLE_STATE_CHANNEL_7 | Sample state register channel 7. |
| ADC_SAMPLE_STATE_CHANNEL_8 | Sample state register channel 8. |
| ADC_SAMPLE_STATE_CHANNEL_9 | Sample state register channel 9. |
| ADC_SAMPLE_STATE_CHANNEL_10 | Sample state register channel 10. |
| ADC_SAMPLE_STATE_CHANNEL_11 | Sample state register channel 11. |
| ADC_SAMPLE_STATE_CHANNEL_12 | Sample state register channel 12. |
| ADC_SAMPLE_STATE_CHANNEL_13 | Sample state register channel 13. |
| ADC_SAMPLE_STATE_CHANNEL_14 | Sample state register channel 14. |
| ADC_SAMPLE_STATE_CHANNEL_15 | Sample state register channel 15. |
| ADC_SAMPLE_STATE_CHANNEL_16_TO_31 | Sample state register channel 16 to 31. |

### ◆ adc_compare_cfg_t

| enum adc_compare_cfg_t |
|---|
| ADC comparison settings |

### ◆ adc_window_b_channel_t

| enum adc_window_b_channel_t |
| --- |
| ADC Window B channel |

### ◆ adc_window_b_mode_t

| enum adc_window_b_mode_t |
| --- |
| ADC Window B comparison mode |

### ◆ adc_group_a_t

| enum adc_group_a_t | |
| --- | --- |
| ADC action for group A interrupts group B scan. This enumeration is used to specify the priority between Group A and B in group mode. | |
| Enumerator | |
| ADC_GROUP_A_PRIORITY_OFF | Group A ignored and does not interrupt ongoing group B scan. |
| ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER | Group A interrupts Group B(single scan) which restarts at next Group B trigger. |
| ADC_GROUP_A_GROUP_B_RESTART_SCAN | Group A interrupts Group B(single scan) which restarts immediately after Group A scan is complete. |
| ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN | Group A interrupts Group B(continuous scan) which continues scanning without a new Group B trigger. |

### ◆ adc_double_trigger_t

| enum adc_double_trigger_t | |
|---|---|
| ADC double-trigger mode definitions | |
| Enumerator | |
| ADC_DOUBLE_TRIGGER_DISABLED | Double-triggering disabled. |
| ADC_DOUBLE_TRIGGER_ENABLED | Double-triggering enabled. |
| ADC_DOUBLE_TRIGGER_ENABLED_EXTENDED | Double-triggering enabled on both ADC ELC events. |

### ◆ adc_start_source_t

| enum adc_start_source_t | |
|---|---|
| ADC Trigger synchronous start source Note: not all sources are available for all MCUs or channels. See User Manual for more information. | |
| Enumerator | |
| ADC_START_SOURCE_DISABLED | ELC/GPT Start source disabled (For use with software start) |
| ADC_START_SOURCE_ASYNC_EXTERNAL | External Trigger Input. |
| ADC_START_SOURCE_ELC_AD0 | ELC_AD0 (Converter 0 and Converter 1) |
| ADC_START_SOURCE_ELC_AD1 | ELC_AD1 (Converter 0 and Converter 1) |
| ADC_START_SOURCE_ELC_AD01 | ELC_AD0 and ELC_AD1 (Converter 0) also ELC_AD0 and ELC_AD1 (Converter 1) |
| ADC_START_SOURCE_GPT_A0_A4 | GTADTRA0 (Converter 0) and GTADTRA4 (Converter 1) |
| ADC_START_SOURCE_GPT_B0_B4 | GTADTRB0 (Converter 0) and GTADTRB4 (Converter 1) |
| ADC_START_SOURCE_GPT_A1_A5 | GTADTRA1 (Converter 0) and GTADTRB5 (Converter 1) |
| ADC_START_SOURCE_GPT_B1_B5 | GTADTRB1 (Converter 0) and GTADTRB5 (Converter 1) |
| ADC_START_SOURCE_GPT_A2_A6 | GTADTRA2 (Converter 0) and GTADTRA6 (Converter 1) |

| | |
|---|---|
| ADC_START_SOURCE_GPT_B2_B6 | GTADTRB2 (Converter 0) and GTADTRB6 (Converter 1) |
| ADC_START_SOURCE_GPT_A3_A7 | GTADTRA3 (Converter 0) and GTADTRA7 (Converter 1) |
| ADC_START_SOURCE_GPT_B3_B7 | GTADTRB3 (Converter 0) and GTADTRB7 (Converter 1) |
| ADC_START_SOURCE_GPT_AB0_AB4 | GTADTRA/B0 (Converter 0) and GTADTRA/B4 (Converter 1) |
| ADC_START_SOURCE_GPT_AB1_AB5 | GTADTRA/B1 (Converter 0) and GTADTRA/B5 (Converter 1) |
| ADC_START_SOURCE_GPT_AB2_AB6 | GTADTRA/B2 (Converter 0) and GTADTRA/B6 (Converter 1) |
| ADC_START_SOURCE_GPT_AB3_AB7 | GTADTRA/B3 (Converter 0) and GTADTRA/B7 (Converter 1) |

### Function Documentation

#### ◆ R_ADC_Open()

fsp_err_t R_ADC_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt is enabled, the function registers a callback function pointer for notifying the user whenever a scan has completed.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is ready for use. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |
| FSP_ERR_IRQ_BSP_DISABLED | A callback is provided, but the interrupt is not enabled. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested unit does not exist on this MCU. |
| FSP_ERR_INVALID_HW_CONDITION | The ADC clock must be at least 1 MHz |

### ◆ R_ADC_ScanCfg()

fsp_err_t R_ADC_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_channel_cfg* )

Configures the ADC scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_channel_cfg_t to p_channel_cfg.

***Note***

*This starts group B scans if adc_channel_cfg_t::priority_group_a is set to ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel specific settings applied. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_CallbackSet()

fsp_err_t R_ADC_CallbackSet ( adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_ADC_ScanStart()

fsp_err_t R_ADC_ScanStart ( adc_ctrl_t * *p_ctrl*)

Starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_Open call. If the unit was configured for ELC or external hardware triggering, then this function allows the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself. If the unit was configured for software triggering, then this function starts the software triggered scan.

**Precondition**

Call R_ADC_ScanCfg after R_ADC_Open before starting a scan.
On MCUs that support calibration, call R_ADC_Calibrate and wait for calibration to complete before starting a scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started (software trigger) or hardware triggers enabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |
| FSP_ERR_IN_USE | Another scan is still in progress (software trigger). |

### ◆ R_ADC_ScanGroupStart()

fsp_err_t R_ADC_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id* )

adc_api_t::scanStart is not supported on the ADCH. Use scanStart instead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_ADC_ScanStop()

fsp_err_t R_ADC_ScanStop ( adc_ctrl_t * *p_ctrl*)

Stops the software scan or disables the unit from being triggered by the hardware trigger (ELC or external) based on what type of trigger the unit was configured for in the R_ADC_Open function. Stopping a hardware triggered scan via this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software triggered scan aborts an ongoing scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped (software trigger) or hardware triggers disabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |

### ◆ R_ADC_StatusGet()

fsp_err_t R_ADC_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module status stored in the provided pointer p_status |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Read()

fsp_err_t R_ADC_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads conversion results from a single channel or sensor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |

### ◆ R_ADC_Read32()

fsp_err_t R_ADC_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads conversion results from a single channel or sensor register into a 32-bit result.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |

### ◆ R_ADC_SampleStateCountSet()

fsp_err_t R_ADC_SampleStateCountSet ( adc_ctrl_t * *p_ctrl*, adc_sample_state_t * *p_sample* )

Sets the sample state count for individual channels. This only needs to be set for special use cases. Normally, use the default values out of reset.

***Note***

*The sample states for the temperature and voltage sensor are set in R_ADC_ScanCfg.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Sample state count updated. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_InfoGet()

fsp_err_t R_ADC_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

Returns the address of the lowest number configured channel and the total number of bytes to be read in order to read the results of the configured channels and return the ELC Event name. If no channels are configured, then a length of 0 is returned.

Also provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

***Note***

*In group mode, information is returned for group A only. Calculating information for group B is not currently supported.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Close()

fsp_err_t R_ADC_Close ( adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_Calibrate()

fsp_err_t R_ADC_Calibrate ( adc_ctrl_t *const *p_ctrl*, void const * *p_extend* )

Initiates calibration of the ADC on MCUs that require calibration. This function must be called before starting a scan on MCUs that require calibration.

Calibration is complete when the callback is called with ADC_EVENT_CALIBRATION_COMPLETE or when R_ADC_StatusGet returns ADC_STATUS_IDLE. See figure "Software flow and operation example of calibration operation." in the ADC section of the relevant hardware manual.

ADC calibration time: 12 PCLKB + 774,930 ADCLK. See table "Required calibration time (shown as the number of ADCLK and PCLKB cycles)" in the ADC section of the relevant hardware manual. The lowest supported ADCLK is 1MHz.

Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. This wait could take up to 780 milliseconds for a 1 MHz PCLKD (ADCLK).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the instance control structure |
| [in] | p_extend | Unused argument. Pass NULL. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calibration successfully initiated. |
| FSP_ERR_INVALID_HW_CONDITION | A scan is in progress or hardware triggers are enabled. |
| FSP_ERR_UNSUPPORTED | Calibration not supported on this MCU. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_OffsetSet()

fsp_err_t R_ADC_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset* )

adc_api_t::offsetSet is not supported on the ADC.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

## 5.2.1.2 ADC (r_adc_b)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_ADC_B_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ADC_B_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_scan_cfg) |
| fsp_err_t | R_ADC_B_CallbackSet (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_ADC_B_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_B_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask) |
| fsp_err_t | R_ADC_B_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_B_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_ADC_B_Read (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint16_t *const p_data) |
| fsp_err_t | R_ADC_B_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint32_t *const p_data) |
| fsp_err_t | R_ADC_B_FifoRead (adc_ctrl_t *p_ctrl, adc_group_mask_t const group_mask, adc_b_fifo_read_t *const p_data) |
| fsp_err_t | R_ADC_B_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_ADC_B_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_B_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend) |
| fsp_err_t | R_ADC_B_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset) |

### Detailed Description

Driver for the ADC_B peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The ADC_B module supports the following features:

- 16 bit resolution
- Selectable data format (16, 14, 12, and 10-bit)
- Configurable high-speed and high-accuracy conversion methods
- Configure scans to include:
    - Multiple analog channels
    - Temperature sensor channel
    - Reference Voltage sensor channel
    - Self-Diagnostic channel
- Configurable scan start trigger:
    - Software scan triggers
    - Hardware scan triggers (timer expiration, for example)
    - External scan triggers from the ADTRGn port pins
- Configurable scan modes:
    - Single scan mode, where each trigger starts a single scan
    - Continuous scan mode, where all channels are scanned continuously
    - Synchronous scan mode, where A/D converters operate synchronously
- Variable sampling time
- Self-calibration
- Channel-dedicated sample-and-hold circuits
- Supports adding and averaging converted samples
- Limiter clip function
- User offset adjustment function
- User gain adjustment function
- Built-in FIFO
- Channel-dedicated programmable gain amplifier (PGA):
    - Support single-ended or pseudo-differential input
    - 2.5x to 13.33x gain (1.5x to 5.56x for pseudo-differential inputs)
- Optional callback when scan completes, FIFO data is ready, an error occurrs, or other conditions are triggered.

# Configuration

### Virtual Channels and Scan Groups

A virtual channel is a group of registers that stores the A/D conversion configuration for a single analog pin. Each virtual channel has a number of options including the channel for conversion, settings for conversion, data processing method and so on.

To perform A/D conversion of an analog pin, the channel associated with the pin must first be assigned to a virtual channel. That virtual channel is then assigned to a scan group, which brings together one or more virtual channels to be converted in sequence with a specified conversion unit.

*Note*

*Analog channels may be assigned to more than one virtual channel. However, a virtual channel can be assigned to only one scan group. When performing A/D conversion on one analog channel in different scan groups or when converting a channel several times within the same scan group, assign several virtual channels to one analog channel.*
*To avoid data being overwritten, when converting a channel multiple times within the same scan group use R_ADC_B_FifoRead() instead of R_ADC_B_Read().*

### Configuring a Scan

To perform A/D conversion of a scan group the following should be configured:

- Assign the analog channel for conversion to a virtual channel.
- Assign the virtual channel to a scan group.
- Assign the scan group to an A/D Converter.

*Note*

*Up to 8 virtual channels can be assigned to a scan group. If more than 8 channels are assigned to a group, only the lowest 8 will be targeted for A/D conversion.*

### Build Time Configurations for r_adc_b

The following build time configurations are defined in fsp_cfg/r_adc_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | bsp | If selected, code for parameter checking is included in the build. |

*Note*

*The instance configurations available in this driver are too numerous to list here. Please refer to the RA Configuration editor in e² studio for further details.*

### Clock Configuration

The ADC_B conversion clock source may be configured to use PCLKC, PCLKA, or GPT with a selectable division ratio. The ADC_B clock may operate between 25 MHz at a minimum and 60 MHz at a maximum.

### Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC_B.

ADTRG0 and ADTRG1 can be used to start scans with an external trigger. When external triggers are used, ADC_B scans begin on the falling edge of the ADTRG pin.

## Usage Notes

### Limitations

Developers should be aware of the following limitations when using R_ADC_B:

- Except for Group Priority Operation, if ADC0 or ADC1 are currently performing an A/D conversion operation, attempting to start another scan group that uses the same A/D converter will be ignored. This also applies to starting multiple groups at one time. When Group Priority Operation is not enabled, only the lowest numbered group will be started (for each ADC converter), other groups will be ignored.

### Self-Calibration

Calibration is required to use this peripheral; call R_ADC_B_Calibrate() after R_ADC_B_ScanCfg() and prior to any other function. Self-Calibration should be performed any time ADC operating characteristics are modified, including after reset, releasing module-stop, when returning from software standby or deep software standby mode, each time the ADC ScanCfg function is called, and

any time ADC related clocks are updated.

*Note*

*Self-calibration is a non-blocking operation. The application should wait for an ADC_EVENT_CALIBRATION_COMPLETE callback before using other ADC_B functionality. The self-calibration process will disable hardware triggers that were previously enabled.*

### ADC_B Operation Modes

The driver supports two primary operational modes, single-scan and continuous-scan. In single scan mode, one or more groups are scanned once per trigger. In continuous scan mode, one or more groups are started with a single trigger. Scans continue until R_ADC_B_ScanStop() is called. In each mode, analog groups and virtual channels are converted in ascending order.

### Single Scan Mode

- Assign any selected analog input or analog channel to any scan group, and convert the selected analog input only once per scan group for each start condition.
- By selecting the scan start conditions for each scan group individually, A/D conversion for each scan group can be started at different times.

### Continuous Scan Mode

- Assign any selected analog input or analog channel to any scan group and repeat A/D conversion in scan group units after the first start condition. Conversion continues until the driver is closed.

### Background Continuous Scan Mode

- Assign any selected analog input or analog channel of the extended analog function to any scan group and repeat A/D conversion in scan group units after the first start condition. The A/D conversion is performed in the background until the driver is closed. In Background Continuous Scan Mode, if the A/D conversion data will be acquired at the point a start condition is entered.

*Note*

*Background Continuous Scan Mode is only available for Hybrid Mode*

### Synchronous-Scan Mode

When synchronous operation is enabled all A/D conversions are guaranteed to begin and end based on a user-configured period. When both conversion units are selected for synchronous scan they run from the same period, allowing for consistent timing of simultaneous conversions. Consult "Synchronous Operation" in the A/D Converter section of the relevant hardware manual.

### ADC_B Conversion Methods

*Note*

*The ADC peripheral supports specific high-speed, high-precision, and normal-precision channels. See Table "A/D conversion characteristics" in the A/D Converter section of the relevant hardware manual for details about what conversion methods are supported by specific physical channels.*

### Successive Approximation Register (SAR) Mode

- A/D Converter samples the signal source once, and convert by Successive Approximation Register method.
- Fast A/D conversion
- Up to 8 channels per 1 scan group.
- Support only single-ended input (excludeing Self-diagnosis function)
- Requires Digital Filter Selection to be disabled

### Oversampling Mode

- A/D Converter oversamples the signal source, and converts analog to digital by Noise Shaping Successive Approximation Register method.
- High-accuracy A/D conversion
- Support up to 8 channels per 1 scan group
- Support single-ended input and differential input
- Requires Digital Filter Selection to be enabled

### Hybrid Mode

- A/D Converter oversamples the signal source, and converts analog to digital by Noise Shaping Successive Approximation Register method.
- Background Continuous Scan Mode operation enables both high-precision A/D conversion and fast conversion.
- Support up to 4 channels per 1 scan group
- Support single-ended input and differential input
- Requires Digital Filter Selection to be enabled

### Digital Filter Selection

The following characteristics of the digital filter for the A/D converter unit may be selected:

- Sinc3 Filter (Over Sampling = 8)
- Minimum Phase Filter (Group Delay < 5)

*Note*

*The digital filter must be disabled for ADC units configured to use Successive Approximation Register (SAR) Mode*
*The digital filter must be enabled for ADC units configured to use Oversampling Mode or Hybrid Mode*

### Sample-and-Hold

Enabling sample-and-hold on one or more channels instructs the ADC to perform sampling on all channels as soon as a group scan is started. Internal circuitry holds the sampled voltages until the conversion unit is ready.

*Note*

*Each sample-and-hold unit is connected to two analog channels (0/1, 2/3 etc). When this function is enabled on both members of a pair only one of the two may be scanned at a time.*
*Enabling sample-and-hold functionality reduces maximum scan frequency because the sample hold time is added to each scan. Refer to the hardware manual for details on the sample-and-hold time. Consult section "ADC Characteristics" of the relevant hardware manual for details.*
*If you use the channel-dedicated sample-and-hold circuits in Hybrid mode, the virtual channels and the scan groups are constrained. A dummy channel must be configured as last within the group. The A/D conversion data of the channel used as the dummy conversion channel is not guaranteed. See Operation in Hybrid Mode with Channel-dedicated Sample-and-hold Circuit section in the User Manual for more information.*

### Self-Diagnosis

ADC_B has a built-in self-diagnosis function that can be used to confirm the unit is working correctly. One of three self-diagnosis voltages can be converted and compared to reference values. A self-diagnosis conversion produces a signed data value indicating the ideal A/D converter result. Reference values for 16-bit data format are shown below.

| Self-diagnosis mode | Expected reference data | Accuracy Error |
|---|---|---|
| Self-diagnosis mode 1 | 0x0000 | A/D conversion result becomes +1 or more when a positive accuracy error occurs and -1 or less when a negative accuracy error occurs. |
| Self-diagnosis mode 1 | 0x8000 | A/D conversion result becomes greater than or equal to reference data when an accuracy error occurs. |
| Self-diagnosis mode 1 | 0x7FFF | A/D conversion result becomes less than or equal to reference data when an accuracy error occurs. |

### Add/Average Function

The ADC can be configured to automatically add or average a number of conversions into a single result. When enabled only the result of the operation is returned.

*Note*

*When the A/D-converted value addition/average function is used, overflow of conversion data may occur. However, in certain conditions overflow may not be detected. See the A/D Conversion Overflow section below for details.*

### Data Format

The A/D converter in this peripheral has a resolution of 16 bits. When 14-bit or 12-bit data format is selected, the lower 2 or 4 bits (respectively) of the A/D conversion result are extended for data processing, error calibration (Self-Calibration), gain/offset adjustment and the averaging function before rounding is applied.

### Limiter Clip

The limit clipping function allows for setting upper and lower bounds on converted data. When the A/D conversion data exceeds the specified upper limit value, it is clipped to the upper limit value. If the A/D conversion data falls below the specified lower limit, it is clipped to the lower limit value.

The upper and lower limits are set in one of eight table entries. To perform limit clip functionality, each virtual channel may (optionally) have one of these entries assigned. Interrupts may be enabled for when limiter clip conditions are triggered.

*Note*

*When 14-/12-/10-bit is selected as the data length of the A/D conversion data, the lower bits are cut based on the data-format selection. When 16-bit format is selected, the data length is not rounded.*

### Using the Temperature Sensor with the ADC_B

The ADC_B HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula:

T = (Vs - V1)/slope + T1

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- slope: Temperature gradient of the temperature sensor (V/degrees C), given as (V2 - V1) / (T2 - T1)
    - T2: Temperature at the experimental measurement of another point (degrees C)
    - V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)

*Note*

*The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.*

### User Offset and Gain

The user offset adjustment function adds or subtracts a constant value to or from the A/D conversion data. Virtual channels select an offset from a table of values specified by the user.

The user gain adjustment function multiplies the A/D conversion data by an arbitrary coefficient value. As with offset adjustment, virtual channels may select a gain value from a table specified by the user.

*Note*

*When the offset or gain adjusting functions are used overflow of A/D conversion data may occur. See the A/D Conversion Overflow section below for details.*
*When 14-/12-/10-bit is selected as the data length of converted data the lower bits of offset values are cut based on the data-format selection.*

### FIFO

The FIFOs consist of 8 stages and can hold up to 8 A/D conversion data. One FIFO is implemented for each scan group. Each FIFO acts as a ring buffer and data will be lost if the FIFO is not read as needed. Interrupts may be enabled for a specific data storage threshold and on overrun.

### Programmable Gain Amplifier

ADC has built-in Programmable Gain Amplifier (PGA). The PGA amplifies an external analog input signal and outputs it to A/D converter, Channel-dedicated sample-and-hold circuit, and High-Speed Analog Comparator (ACMPHS). PGA units are channel specific and utilize two analog input pins per unit. Please refer to the RA Configuration editor in e² studio or the hardware manual for further details.

*Note*

*When PGA is used, the analog input pin assigned to PGAVSS pin cannot be input to A/D conversion or Channel-dedicated sample-and-hold circuit.*

### Single-ended input

In single-ended input mode, PGA amplifies the input from PGAIN pin with the specified gain, between ×2.000 to ×13.333. When operating the PGA in single-ended input mode, PGAIN should be connected to the signal source and PGAVSS should be connected to the analog ground (AVSS0). The input voltage to PGAIN must not exceed the range specified in the Electrical Characteristics.

#### Pseudo-differential input

In pseudo differential input mode, PGA amplifies the difference between PGAIN pin and PGAVSS pin with the specified gain and output the voltage obtained by adding the offset of 0.5 * AVcc. Settable gains are ×1.500, ×2.333, ×4.000, and ×5.667.

When operating the PGA in Pseudo Differential Input Mode, PGAIN should be connected to the signal source, and PGAVSS should be connected to the reference ground of the signal source. The inputs to PGAIN and PGAVSS pins must not exceed the range specified in the Electrical Characteristics.

#### When Interrupts Are Not Enabled

Interrupts are enabled by default. If scan-complete interrupts are disabled,@ ref R_ADC_B_StatusGet() can be used to poll the ADC_B driver to determine when the scan has completed. R_ADC_B_Read() is used to access the converted ADC_B result.

#### A/D Conversion Overflow

A/D conversion overflow is detected when converted data exceeds the range that can be handled in the specified data format. When overflow occurs, data is restricted to the upper or lower limit value of the specified data format. Overflow is detected in the following cases:

- When the input to the A/D converter exceeds VREFH0 or falls below VREFL0
- When overflow occurs by the internal processing (calculation) for the A/D conversion data due to the following: – Gain Error and Offset Error Calibration – User Gain/Offset adjustment function – When using A/D-Converted Value Addition/Averaging Function – Data Formatting Process

## Examples

#### Basic Example

This is a basic example of minimal use of the ADC_B in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_B_ScanCfg. */
extern const adc_b_scan_cfg_t g_adc_b0_scan_cfg;
void adc_b_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = R_ADC_B_Open(&g_adc_b0_ctrl, &g_adc_b0_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
    err = R_ADC_B_ScanCfg(&g_adc_b0_ctrl, &g_adc_b0_scan_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_B_Calibrate(&g_adc_b0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
 /* Wait for calibration to complete */
 adc_status_t status = {.state = ADC_STATE_CALIBRATION_IN_PROGRESS};
 while ((ADC_STATE_IDLE != status.state) &&
          (FSP_SUCCESS == err))
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       err = R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
    }
    assert(FSP_SUCCESS == err);
 /* Start one or more scan groups by calling R_ADC_B_ScanGroupStart(). Alternatively,
all scan groups may be started
  * by calling R_ADC_B_ScanCfg(). */
    (void) R_ADC_B_ScanGroupStart(&g_adc_b0_ctrl, ADC_GROUP_MASK_0);
 /* Wait for conversion to complete. */
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_IDLE != status.state)
    {
       (void) R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t channel_0_conversion_result;
    err = R_ADC_B_Read(&g_adc_b0_ctrl, ADC_CHANNEL_0, &channel_0_conversion_result);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | adc_b_fifo_data_t |

| | |
|---|---|
| struct | adc_b_fifo_read_t |
| struct | adc_b_group_cfg_t |
| struct | adc_b_scan_cfg_t |
| struct | adc_b_isr_cfg_t |
| struct | adc_b_extended_cfg_t |
| struct | adc_b_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | adc_b_clock_divider_t |
| enum | adc_b_converter_mode_t |
| enum | adc_b_conversion_method_t |
| enum | adc_b_data_format_t |
| enum | adc_b_limit_clip_table_id_t |
| enum | adc_b_unit_id_t |
| enum | adc_b_unit_mask_t |
| enum | adc_b_add_avg_mode_t |
| enum | adc_b_add_avg_count_t |
| enum | adc_b_gpt_trigger_t |
| enum | adc_b_external_trigger_t |
| enum | adc_b_self_diagnosis_mode_t |
| enum | adc_b_sample_and_hold_mask_t |
| enum | adc_b_pga_gain_t |
| enum | adc_b_digital_filter_selection_t |
| enum | adc_b_sampling_state_table_id_t |
| enum | adc_b_user_gain_table_id_t |
| enum | adc_b_user_offset_table_selection_id_t |

## Data Structure Documentation

### ◆ adc_b_fifo_data_t

| struct adc_b_fifo_data_t | | |
|---|---|---|
| ADC FIFO data type | | |
| Data Fields | | |
| uint32_t | data: 16 | Conversion Data. |
| uint32_t | __pad0__: 8 | |
| adc_channel_t | physical_channel: 7 | Channel number for data. |
| uint32_t | err: 1 | Error bit. |

### ◆ adc_b_fifo_read_t

| struct adc_b_fifo_read_t | | |
|---|---|---|
| ADC FIFO Read data structure | | |
| Data Fields | | |
| uint8_t | count | Number of valid FIFO data read. |
| adc_b_fifo_data_t | fifo_data[8] | FIFO data. |

### ◆ adc_b_group_cfg_t

| struct adc_b_group_cfg_t | | |
|---|---|---|
| ADC Group configuration data | | |
| Data Fields | | |
| adc_group_id_t | scan_group_id | Scan Group ID. |
| adc_b_unit_id_t | converter_selection | Converter selection. |
| bool | scan_group_enable | Scan Group enable state. |
| uint8_t | virtual_channel_count | Virtual Channel count. |
| bool | scan_end_interrupt_enable | Scan End Interrupt enable. |
| adc_b_external_trigger_t | external_trigger_enable_mask | External Trigger mask. |
| elc_peripheral_t | elc_trigger_enable_mask | ELC Trigger mask. |
| adc_b_gpt_trigger_t | gpt_trigger_enable_mask | GPT Trigger mask. |
| uint8_t | conversion_start_delay | Conversion start delay. |
| uint32_t | self_diagnosis_mask | Self-Diagnosis register data. |
| bool | limit_clip_interrupt_enable | Limiter Clip interrupt enable. |
| adc_b_virtual_channel_cfg_t ** | p_virtual_channels | Pointer to virtual channel configuration array of size virtual_channel_count. |

### ◆ adc_b_scan_cfg_t

| struct adc_b_scan_cfg_t | | |
|---|---|---|
| ADC Scan Group configuration | | |
| Data Fields | | |
| uint8_t | group_count | Group Count. |
| adc_b_group_cfg_t ** | p_adc_groups | Pointer to ADC group configuration data. |

### ◆ adc_b_isr_cfg_t

| struct adc_b_isr_cfg_t | | |
|---|---|---|
| ADC ISR configuration structure | | |
| Data Fields | | |
| uint8_t | calibration_end_ipl_adc_0 | Calibration end IPL for A/D converter unit 0. |
| uint8_t | calibration_end_ipl_adc_1 | Calibration end IPL for A/D converter unit 1. |
| uint8_t | conversion_error_ipl_adc_0 | Conversion error IPL for A/D converter unit 0. |
| uint8_t | conversion_error_ipl_adc_1 | Conversion error IPL for A/D converter unit 1. |
| uint8_t | fifo_overflow_ipl | FIFO Overflow IPL. |
| uint8_t | fifo_read_ipl_group_0 | FIFO Read threshold request IPL for Group 0. |
| uint8_t | fifo_read_ipl_group_1 | FIFO Read threshold request IPL for Group 1. |
| uint8_t | fifo_read_ipl_group_2 | FIFO Read threshold request IPL for Group 2. |
| uint8_t | fifo_read_ipl_group_3 | FIFO Read threshold request IPL for Group 3. |
| uint8_t | fifo_read_ipl_group_4 | FIFO Read threshold request IPL for Group 4. |
| uint8_t | fifo_read_ipl_group_5 | FIFO Read threshold request IPL for Group 5 (available on some MCUs) |
| uint8_t | fifo_read_ipl_group_6 | FIFO Read threshold request IPL for Group 6 (available on some MCUs) |
| uint8_t | fifo_read_ipl_group_7 | FIFO Read threshold request IPL for Group 7 (available on some MCUs) |
| uint8_t | fifo_read_ipl_group_8 | FIFO Read threshold request IPL |

| | | |
|---|---|---|
| | | for Group 8 (available on some MCUs) |
| uint8_t | fifo_read_ipl_group_5678 | FIFO Read threshold request IPL for Groups 5, 6, 7, and 8 (available on some MCUs) |
| uint8_t | limit_clip_ipl | Limiter Clip IPL. |
| uint8_t | overflow_error_ipl_adc_0 | Overflow error IPL for A/D converter unit 0. |
| uint8_t | overflow_error_ipl_adc_1 | Overflow error IPL for A/D converter unit 1. |
| uint8_t | scan_end_ipl_group_0 | Scan End IPL for A/D Group 0. |
| uint8_t | scan_end_ipl_group_1 | Scan End IPL for A/D Group 1. |
| uint8_t | scan_end_ipl_group_2 | Scan End IPL for A/D Group 2. |
| uint8_t | scan_end_ipl_group_3 | Scan End IPL for A/D Group 3. |
| uint8_t | scan_end_ipl_group_4 | Scan End IPL for A/D Group 4. |
| uint8_t | scan_end_ipl_group_5 | Scan End IPL for A/D Group 5 (available on some MCUs) |
| uint8_t | scan_end_ipl_group_6 | Scan End IPL for A/D Group 6 (available on some MCUs) |
| uint8_t | scan_end_ipl_group_7 | Scan End IPL for A/D Group 7 (available on some MCUs) |
| uint8_t | scan_end_ipl_group_8 | Scan End IPL for A/D Group 8 (available on some MCUs) |
| uint8_t | scan_end_ipl_group_5678 | Scan End IRQ for A/D Groups 5, 6, 7, and 8 (available on some MCUs) |
| IRQn_Type | calibration_end_irq_adc_0 | Calibration end IRQ for A/D converter unit 0. |
| IRQn_Type | calibration_end_irq_adc_1 | Calibration end IRQ for A/D converter unit 1. |
| IRQn_Type | conversion_error_irq_adc_0 | Conversion error IRQ for A/D converter unit 0. |
| IRQn_Type | conversion_error_irq_adc_1 | Conversion error IRQ for A/D converter unit 1. |
| IRQn_Type | fifo_overflow_irq | FIFO Overflow IRQ. |
| IRQn_Type | fifo_read_irq_group_0 | FIFO Read threshold request IRQ for Group 0. |
| IRQn_Type | fifo_read_irq_group_1 | FIFO Read threshold request IRQ for Group 1. |
| IRQn_Type | fifo_read_irq_group_2 | FIFO Read threshold request IRQ for Group 2. |

| | | |
|---|---|---|
| IRQn_Type | fifo_read_irq_group_3 | FIFO Read threshold request IRQ for Group 3. |
| IRQn_Type | fifo_read_irq_group_4 | FIFO Read threshold request IRQ for Group 4. |
| IRQn_Type | fifo_read_irq_group_5 | FIFO Read threshold request IRQ for Group 5. |
| IRQn_Type | fifo_read_irq_group_6 | FIFO Read threshold request IRQ for Group 6. |
| IRQn_Type | fifo_read_irq_group_7 | FIFO Read threshold request IRQ for Group 7. |
| IRQn_Type | fifo_read_irq_group_8 | FIFO Read threshold request IRQ for Group 8. |
| IRQn_Type | fifo_read_irq_group_5678 | FIFO Read threshold request IRQ for Groups 5, 6, 7, and 8. |
| IRQn_Type | limit_clip_irq | Limiter Clip IRQ. |
| IRQn_Type | overflow_error_irq_adc_0 | Overflow error IRQ for A/D converter unit 0. |
| IRQn_Type | overflow_error_irq_adc_1 | Overflow error IRQ for A/D converter unit 1. |
| IRQn_Type | scan_end_irq_group_0 | Scan End IRQ for A/D Group 0. |
| IRQn_Type | scan_end_irq_group_1 | Scan End IRQ for A/D Group 1. |
| IRQn_Type | scan_end_irq_group_2 | Scan End IRQ for A/D Group 2. |
| IRQn_Type | scan_end_irq_group_3 | Scan End IRQ for A/D Group 3. |
| IRQn_Type | scan_end_irq_group_4 | Scan End IRQ for A/D Group 4. |
| IRQn_Type | scan_end_irq_group_5 | Scan End IRQ for A/D Group 5. |
| IRQn_Type | scan_end_irq_group_6 | Scan End IRQ for A/D Group 6. |
| IRQn_Type | scan_end_irq_group_7 | Scan End IRQ for A/D Group 7. |
| IRQn_Type | scan_end_irq_group_8 | Scan End IRQ for A/D Group 8. |
| IRQn_Type | scan_end_irq_group_5678 | Scan End IRQ for A/D Groups 5, 6, 7, and 8. |

### ◆ adc_b_extended_cfg_t

| struct adc_b_extended_cfg_t | | |
|---|---|---|
| ADC extended configuration data | | |
| Data Fields | | |
| adc_b_pga_gain_t | pga_gain[4] | PGA Gain selection. |
| union adc_b_extended_cfg_t | __unnamed__ | |
| union adc_b_extended_cfg_t | adc_filter_selection[2] | |

| | | |
|---|---|---|
| union adc_b_extended_cfg_t | __unnamed__ | |
| union adc_b_extended_cfg_t | __unnamed__ | |
| union adc_b_extended_cfg_t | __unnamed__ | |
| uint32_t | scan_group_enable | Scan Group enable register data. |
| union adc_b_extended_cfg_t | __unnamed__ | |
| union adc_b_extended_cfg_t | __unnamed__ | |
| uint16_t | fifo_interrupt_enable_mask | FIFO interrupt enable register data. |
| union adc_b_extended_cfg_t | __unnamed__ | |
| union adc_b_extended_cfg_t | __unnamed__ | |
| uint32_t | calibration_adc_state | Calibration State register data. |
| uint32_t | calibration_sample_and_hold | Calibration Sample and Hold register data. |
| const adc_b_isr_cfg_t * | p_isr_cfg | Pointer to ISR configuration. |
| union adc_b_extended_cfg_t | __unnamed__ | |
| uint8_t | sample_and_hold_enable_mask | Sample and Hold enable register data. |
| uint32_t | sample_and_hold_config_012 | Sample and Hold configuration register data. |
| uint32_t | sample_and_hold_config_456 | Sample and Hold configuration register data. |
| uint32_t | conversion_state | ADC 0/1 Successive Approximation Time Configuration. |
| int32_t | user_offset_tables[8] | User Offset Table register data. |
| uint32_t | user_gain_tables[8] | User Gain Table register data. |
| uint32_t | limiter_clip_interrupt_enable_mask | Limiter clip interrupt enable register data. |
| uint32_t | limiter_clip_tables[8] | Limiter clip Table register data. |

### ◆ adc_b_instance_ctrl_t

| struct adc_b_instance_ctrl_t | |
|---|---|
| ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). | |
| **Data Fields** | |
| adc_b_converter_state_t | adc_state |
| | ADC 0 converter State. |

| | |
|---|---|
| | |
| uint32_t | cached_adtrgenr |
| | Cached conversion peripheral trigger bits, used when starting and stopping scans. |
| | |
| uint32_t | cached_adsystr |
| | Cached conversion software start bits, used when starting and stopping scans. |
| | |
| uint32_t | trigger_disable_wait_cycles |
| | ADC clock cycles required to wait after disabling trigger input. |
| | |
| adc_cfg_t const * | p_cfg |
| | Boolean to verify that the Unit has been initialized. |
| | |
| void(* | p_callback )(adc_callback_args_t *) |
| | Pointer to callback that is called when an adc_b_event_t occurs. |
| | |
| adc_callback_args_t * | p_callback_memory |
| | Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| uint32_t | initialized |
| | Initialized status of ADC_B. |
| | |
| uint32_t | opened |
| | Open status of ADC_B. |

## Enumeration Type Documentation

### ◆ adc_b_clock_divider_t

| enum adc_b_clock_divider_t | |
|---|---|
| ADC clock divider selection | |
| Enumerator | |
| ADC_B_CLOCK_DIV_1 | ADC Clock Division 1/1. |
| ADC_B_CLOCK_DIV_2 | ADC Clock Division 1/2. |
| ADC_B_CLOCK_DIV_3 | ADC Clock Division 1/3. |
| ADC_B_CLOCK_DIV_4 | ADC Clock Division 1/4. |
| ADC_B_CLOCK_DIV_5 | ADC Clock Division 1/5. |
| ADC_B_CLOCK_DIV_6 | ADC Clock Division 1/6. |
| ADC_B_CLOCK_DIV_7 | ADC Clock Division 1/7. |
| ADC_B_CLOCK_DIV_8 | ADC Clock Division 1/8. |

### ◆ adc_b_converter_mode_t

| enum adc_b_converter_mode_t | |
|---|---|
| ADC_B Conversion Mode | |
| Enumerator | |
| ADC_B_CONVERTER_MODE_SINGLE_SCAN | Single scan mode. |
| ADC_B_CONVERTER_MODE_CONTINUOUS_SCAN | Continuous scan mode. |
| ADC_B_CONVERTER_MODE_BACKGROUND_SCAN | Background continuous scan mode (Valid for Hybrid mode only) |

### ◆ adc_b_conversion_method_t

| enum adc_b_conversion_method_t | |
|---|---|
| ADC_B Conversion Method | |
| Enumerator | |
| ADC_B_CONVERSION_METHOD_SAR | SAR conversion method. |
| ADC_B_CONVERSION_METHOD_OVERSAMPLE | Oversampling conversion method. |
| ADC_B_CONVERSION_METHOD_HYBRID | Hybrid conversion method. |

### ◆ adc_b_data_format_t

| enum adc_b_data_format_t | |
|---|---|
| ADC_B data data format definitions | |
| Enumerator | |
| ADC_B_DATA_FORMAT_16_BIT | 16 bit adc_b data format |
| ADC_B_DATA_FORMAT_14_BIT | 14 bit adc_b data format |
| ADC_B_DATA_FORMAT_12_BIT | 12 bit adc_b data format |
| ADC_B_DATA_FORMAT_10_BIT | 10 bit adc_b data format |

### ◆ adc_b_limit_clip_table_id_t

| enum adc_b_limit_clip_table_id_t | |
|---|---|
| ADC limiter clipping table id selection options | |
| Enumerator | |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_NONE | Limiter Clip Disabled. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_0 | Limiter Clip Table 0. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_1 | Limiter Clip Table 1. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_2 | Limiter Clip Table 2. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_3 | Limiter Clip Table 3. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_4 | Limiter Clip Table 4. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_5 | Limiter Clip Table 5. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_6 | Limiter Clip Table 6. |
| ADC_B_LIMIT_CLIP_TABLE_SELECTION_7 | Limiter Clip Table 7. |

### ◆ adc_b_unit_id_t

| enum adc_b_unit_id_t | |
|---|---|
| ADC unit selection options | |
| Enumerator | |
| ADC_B_UNIT_ID_0 | ADC Unit ID 0. |
| ADC_B_UNIT_ID_1 | ADC Unit ID 1. |

### ◆ adc_b_unit_mask_t

| enum adc_b_unit_mask_t | |
|---|---|
| ADC unit selection options | |
| Enumerator | |
| ADC_B_UNIT_MASK_0 | ADC Unit Mask 0. |
| ADC_B_UNIT_MASK_1 | ADC Unit Mask 1. |
| ADC_B_UNIT_MASK_UNDEFINED | ADC Unit Mask Unknown. |

### ◆ adc_b_add_avg_mode_t

| enum adc_b_add_avg_mode_t | |
|---|---|
| ADC data sample addition and averaging options | |
| Enumerator | |
| ADC_B_ADD_AVERAGE_OFF | Add/Average turned off for channels/sensors. |
| ADC_B_ADD_AVERAGE_ADDITION_ENABLE | Addition Mode Enabled. |
| ADC_B_ADD_AVERAGE_AVERAGE_ENABLE | Average Mode Enabled. |

### ◆ adc_b_add_avg_count_t

| enum adc_b_add_avg_count_t | |
|---|---|
| ADC data sample addition and averaging options | |
| Enumerator | |
| ADC_B_ADD_AVERAGE_1 | Addition turned off for channels/sensors. |
| ADC_B_ADD_AVERAGE_2 | Add/Average 2 samples. |
| ADC_B_ADD_AVERAGE_4 | Add/Average 4 samples. |
| ADC_B_ADD_AVERAGE_8 | Add/Average 8 samples. |
| ADC_B_ADD_AVERAGE_16 | Add/Average 16 samples. |
| ADC_B_ADD_AVERAGE_32 | Add/Average 32 samples. |
| ADC_B_ADD_AVERAGE_64 | Add/Average 64 samples. |
| ADC_B_ADD_AVERAGE_128 | Add/Average 128 samples. |
| ADC_B_ADD_AVERAGE_256 | Add/Average 256 samples. |
| ADC_B_ADD_AVERAGE_512 | Add/Average 512 samples. |
| ADC_B_ADD_AVERAGE_1024 | Add/Average 1024 samples. |

### ◆ adc_b_gpt_trigger_t

| enum adc_b_gpt_trigger_t | |
|---|---|
| ADC GPT Trigger options | |
| Enumerator | |
| ADC_B_GPT_TRIGGER_NONE | GPT Trigger Disabled. |
| ADC_B_GPT_TRIGGER_A0 | GPT Trigger A0. |
| ADC_B_GPT_TRIGGER_A1 | GPT Trigger A1. |
| ADC_B_GPT_TRIGGER_A2 | GPT Trigger A2. |
| ADC_B_GPT_TRIGGER_A3 | GPT Trigger A3. |
| ADC_B_GPT_TRIGGER_A4 | GPT Trigger A4. |

| | |
|---|---|
| ADC_B_GPT_TRIGGER_A5 | GPT Trigger A5. |
| ADC_B_GPT_TRIGGER_A6 | GPT Trigger A6. |
| ADC_B_GPT_TRIGGER_A7 | GPT Trigger A7. |
| ADC_B_GPT_TRIGGER_A8 | GPT Trigger A8. |
| ADC_B_GPT_TRIGGER_A9 | GPT Trigger A9. |
| ADC_B_GPT_TRIGGER_A10 | GPT Trigger A10. |
| ADC_B_GPT_TRIGGER_A11 | GPT Trigger A11. |
| ADC_B_GPT_TRIGGER_A12 | GPT Trigger A12. |
| ADC_B_GPT_TRIGGER_A13 | GPT Trigger A13. |
| ADC_B_GPT_TRIGGER_B0 | GPT Trigger B0. |
| ADC_B_GPT_TRIGGER_B1 | GPT Trigger B1. |
| ADC_B_GPT_TRIGGER_B2 | GPT Trigger B2. |
| ADC_B_GPT_TRIGGER_B3 | GPT Trigger B3. |
| ADC_B_GPT_TRIGGER_B4 | GPT Trigger B4. |
| ADC_B_GPT_TRIGGER_B5 | GPT Trigger B5. |
| ADC_B_GPT_TRIGGER_B6 | GPT Trigger B6. |
| ADC_B_GPT_TRIGGER_B7 | GPT Trigger B7. |
| ADC_B_GPT_TRIGGER_B8 | GPT Trigger B8. |
| ADC_B_GPT_TRIGGER_B9 | GPT Trigger B9. |
| ADC_B_GPT_TRIGGER_B10 | GPT Trigger B10. |
| ADC_B_GPT_TRIGGER_B11 | GPT Trigger B11. |
| ADC_B_GPT_TRIGGER_B12 | GPT Trigger B12. |
| ADC_B_GPT_TRIGGER_B13 | GPT Trigger B13. |

### ◆ adc_b_external_trigger_t

| enum adc_b_external_trigger_t | |
|---|---|
| ADC External Trigger options | |
| Enumerator | |
| ADC_B_EXTERNAL_TRIGGER_NONE | External Trigger Disabled. |
| ADC_B_EXTERNAL_TRIGGER_ADTRG0 | External Trigger ADTRG0 Selection. |
| ADC_B_EXTERNAL_TRIGGER_ADTRG1 | External Trigger ADTRG1 Selection. |

### ◆ adc_b_self_diagnosis_mode_t

| enum adc_b_self_diagnosis_mode_t | |
|---|---|
| ADC Self-Diagnosis mode options | |
| Enumerator | |
| ADC_B_SELF_DIAGNOSIS_DISABLED | Self-Diagnosis Disabled. |
| ADC_B_SELF_DIAGNOSIS_MODE_1 | Self-Diagnosis Mode 1. |
| ADC_B_SELF_DIAGNOSIS_MODE_2 | Self-Diagnosis Mode 2. |
| ADC_B_SELF_DIAGNOSIS_MODE_3 | Self-Diagnosis Mode 3. |

### ◆ adc_b_sample_and_hold_mask_t

| enum adc_b_sample_and_hold_mask_t | |
|---|---|
| ADC Sample-and-Hold unit enable mask | |
| Enumerator | |
| ADC_B_SAMPLE_AND_HOLD_MASK_NONE | Sample-and-Hold Disabled. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_0 | Sample-and-Hold Unit 0. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_1 | Sample-and-Hold Unit 1. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_2 | Sample-and-Hold Unit 2. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_4 | Sample-and-Hold Unit 3. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_5 | Sample-and-Hold Unit 4. |
| ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_6 | Sample-and-Hold Unit 5. |

### ◆ adc_b_pga_gain_t

| enum adc_b_pga_gain_t | |
|---|---|
| ADC PGA Gain | |
| Enumerator | |
| ADC_B_PGA_GAIN_DISABLED | PGA Gain Disabled. |
| ADC_B_PGA_GAIN_DIFFERENTIAL_1_500 | PGA Gain Setting 1.500. |
| ADC_B_PGA_GAIN_DIFFERENTIAL_2_333 | PGA Gain Setting 2.333. |
| ADC_B_PGA_GAIN_DIFFERENTIAL_4_000 | PGA Gain Setting 4.000. |
| ADC_B_PGA_GAIN_DIFFERENTIAL_5_667 | PGA Gain Setting 5.667. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_2_500 | PGA Gain Setting 2.500. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_2_667 | PGA Gain Setting 2.667. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_2_857 | PGA Gain Setting 2.857. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_3_077 | PGA Gain Setting 3.077. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_3_333 | PGA Gain Setting 3.333. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_3_636 | PGA Gain Setting 3.636. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_4_000 | PGA Gain Setting 4.000. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_4_444 | PGA Gain Setting 4.444. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_5_000 | PGA Gain Setting 5.000. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_5_714 | PGA Gain Setting 5.714. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_6_667 | PGA Gain Setting 6.667. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_8_000 | PGA Gain Setting 8.000. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_10_000 | PGA Gain Setting 10.000. |
| ADC_B_PGA_GAIN_SINGLE_ENDED_13_333 | PGA Gain Setting 13.333. |

### ◆ adc_b_digital_filter_selection_t

| enum adc_b_digital_filter_selection_t | |
|---|---|
| ADC Digital Filter Selection | |
| Enumerator | |
| ADC_B_DIGITAL_FILTER_MODE_SINC3 | Digital filter Sinc3 filter (Oversampling Rate = 8) |
| ADC_B_DIGITAL_FILTER_MODE_PHASE | Digital filter Minimum phase filter (Group delay < 2) |

### ◆ adc_b_sampling_state_table_id_t

| enum adc_b_sampling_state_table_id_t | |
|---|---|
| ADC Sampling State table selection options | |
| Enumerator | |
| ADC_B_SAMPLING_STATE_TABLE_0 | Sampling State Table 0. |
| ADC_B_SAMPLING_STATE_TABLE_1 | Sampling State Table 1. |
| ADC_B_SAMPLING_STATE_TABLE_2 | Sampling State Table 2. |
| ADC_B_SAMPLING_STATE_TABLE_3 | Sampling State Table 3. |
| ADC_B_SAMPLING_STATE_TABLE_4 | Sampling State Table 4. |
| ADC_B_SAMPLING_STATE_TABLE_5 | Sampling State Table 5. |
| ADC_B_SAMPLING_STATE_TABLE_6 | Sampling State Table 6. |
| ADC_B_SAMPLING_STATE_TABLE_7 | Sampling State Table 7. |
| ADC_B_SAMPLING_STATE_TABLE_8 | Sampling State Table 8. |
| ADC_B_SAMPLING_STATE_TABLE_9 | Sampling State Table 9. |
| ADC_B_SAMPLING_STATE_TABLE_10 | Sampling State Table 10. |
| ADC_B_SAMPLING_STATE_TABLE_11 | Sampling State Table 12. |
| ADC_B_SAMPLING_STATE_TABLE_12 | Sampling State Table 13. |
| ADC_B_SAMPLING_STATE_TABLE_13 | Sampling State Table 14. |
| ADC_B_SAMPLING_STATE_TABLE_14 | Sampling State Table 15. |
| ADC_B_SAMPLING_STATE_TABLE_15 | Sampling State Table 16. |

### ◆ adc_b_user_gain_table_id_t

| enum adc_b_user_gain_table_id_t | |
|---|---|
| ADC User Gain table options | |
| Enumerator | |
| ADC_B_USER_GAIN_TABLE_SELECTION_DISABLED | User Gain disabled. |
| ADC_B_USER_GAIN_TABLE_SELECTION_0 | User Gain table 0. |
| ADC_B_USER_GAIN_TABLE_SELECTION_1 | User Gain table 1. |
| ADC_B_USER_GAIN_TABLE_SELECTION_2 | User Gain table 2. |
| ADC_B_USER_GAIN_TABLE_SELECTION_3 | User Gain table 3. |
| ADC_B_USER_GAIN_TABLE_SELECTION_4 | User Gain table 4. |
| ADC_B_USER_GAIN_TABLE_SELECTION_5 | User Gain table 5. |
| ADC_B_USER_GAIN_TABLE_SELECTION_6 | User Gain table 6. |
| ADC_B_USER_GAIN_TABLE_SELECTION_7 | User Gain table 7. |

### ◆ adc_b_user_offset_table_selection_id_t

| enum adc_b_user_offset_table_selection_id_t | |
|---|---|
| ADC User Offset table options | |
| Enumerator | |
| ADC_B_USER_OFFSET_TABLE_SELECTION_DISABLED | User Offset disabled. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_0 | User Offset table 0. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_1 | User Offset table 1. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_2 | User Offset table 2. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_3 | User Offset table 3. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_4 | User Offset table 4. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_5 | User Offset table 5. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_6 | User Offset table 6. |
| ADC_B_USER_OFFSET_TABLE_SELECTION_7 | User Offset table 7. |

## Function Documentation

### ◆ R_ADC_B_Open()

fsp_err_t R_ADC_B_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If provided, the function registers a callback function pointer for notifying the user whenever a scan has completed, error has occurred, FIFO read request is generated, or other ADC interrupt event occurrs. Implements adc_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is ready for use. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |

### ◆ R_ADC_B_ScanCfg()

fsp_err_t R_ADC_B_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_scan_cfg* )

Configures the ADC_B scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_b_scan_cfg_t to p_channel_cfg. Implements adc_api_t::scanCfg.

***Note***

*This starts group B scans if adc_b_scan_cfg_t::priority_group_a is set to ADC_B_GROUP_A_GROUP_B_CONTINUOUS_SCAN.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel specific settings applied. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_INVALID_STATE | Invalid Scan Configuration. |
| FSP_ERR_INVALID_CHANNEL | Invalid configured channel for group converter id. |

### ◆ R_ADC_B_CallbackSet()

fsp_err_t R_ADC_B_CallbackSet ( adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_ADC_B_ScanStart()

fsp_err_t R_ADC_B_ScanStart ( adc_ctrl_t * *p_ctrl*)

Enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_B_ScanCfg call. If the unit was configured for ELC, GPT, or external hardware triggering, then this function allows the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself. If the unit was configured for software triggering, This function was added to this ADC version for compatibility with r_adc driver. For additional flexibility, it is recommended to use R_ADC_B_ScanGroupStart.

**Precondition**

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started (software trigger) or hardware triggers enabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_INVALID_ARGUMENT | No hardware triggers configured for groups. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |
| FSP_ERR_INVALID_STATE | Calibration required. |

### ◆ R_ADC_B_ScanGroupStart()

fsp_err_t R_ADC_B_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_mask* )

Starts a software scan or enables the hardware trigger for a scan depending on how triggers were configured. If the group was configured for ELC, GPT, or external hardware triggering then this function allows the trigger signal to get to the ADC unit. The function itself is not able to control the generation of peripheral triggers. If the unit was configured for software triggering, then this function starts the software triggered scan.

***Note***

*Except for Group Priority Operation, if ADC0 or ADC1 are currently performing an A/D conversion operation, attempting to start another scan group that uses the same A/D converter will be ignored. This also applies to starting multiple groups at one time. When Group Priority Operation is not enabled, only the lowest numbered group will be started (for each ADC converter), other groups will be ignored.*

**Precondition**

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started (software trigger) or hardware triggers enabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_INVALID_ARGUMENT | An invalid group has been provided. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |
| FSP_ERR_INVALID_STATE | Calibration required. |

### ◆ R_ADC_B_ScanStop()

fsp_err_t R_ADC_B_ScanStop ( adc_ctrl_t * *p_ctrl*)

Disables the hardware trigger for a scan and immediately stops all active converters. This function will abortall active conversions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | All scans stopped. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_INVALID_ARGUMENT | No hardware triggers configured for groups. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |

### ◆ R_ADC_B_StatusGet()

fsp_err_t R_ADC_B_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module status stored in the provided pointer p_status |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_B_Read()

fsp_err_t R_ADC_B_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint16_t *const *p_data* )

Reads conversion results from a single channel or sensor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_INVALID_DATA | Accuracy of data cannot be guaranteed. ADC requires calibration or SAR timing settings are irregular. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel provided. |

### ◆ R_ADC_B_Read32()

fsp_err_t R_ADC_B_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint32_t *const *p_data* )

Reads conversion results from a single channel or sensor register into a 32-bit result.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_INVALID_DATA | Accuracy of data cannot be guaranteed. ADC requires calibration or SAR timing settings are irregular. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel provided. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- adc_api_t::read

### ◆ R_ADC_B_FifoRead()

fsp_err_t R_ADC_B_FifoRead ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t const *group_mask*, adc_b_fifo_read_t *const *p_data* )

Reads conversion results from FIFO for the given group mask.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |
| FSP_ERR_INVALID_ARGUMENT | Invalid group provided. |
| FSP_ERR_UNDERFLOW | FIFO empty. |

### ◆ R_ADC_B_InfoGet()

fsp_err_t R_ADC_B_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

Provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Info is read into p_adc_info. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit not initialized. |

### ◆ R_ADC_B_Close()

fsp_err_t R_ADC_B_Close ( adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_B_Calibrate()

fsp_err_t R_ADC_B_Calibrate ( adc_ctrl_t *const *p_ctrl*, void const * *p_extend* )

Initiates calibration of the ADC_B. This function must be called before starting a scan and again whenever ADC_B configuration or state is changed.

***Note***

*Self-calibration is a non-blocking operation. The application must wait for calibration to complete before using other ADC_B functionality.*
*The self-calibration process will disable hardware triggers that were previously enabled.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the instance control structure |
| [in] | p_extend | Unused argument. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calibration successfully initiated. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_ADC_B_OffsetSet()

fsp_err_t R_ADC_B_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset* )

adc_api_t::offsetSet is not supported on the ADC_B.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

## 5.2.1.3 ADC (r_adc_d)

Modules » Analog

**Functions**

| | |
|---|---|
| fsp_err_t | R_ADC_D_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ADC_D_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_channel_cfg) |
| fsp_err_t | R_ADC_D_CallbackSet (adc_ctrl_t *const p_api_ctrl, |

| | |
|---|---|
| | void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_ADC_D_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_D_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id) |
| fsp_err_t | R_ADC_D_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_D_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_ADC_D_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_ADC_D_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_ADC_D_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_ADC_D_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_ADC_D_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend) |
| fsp_err_t | R_ADC_D_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset) |
| fsp_err_t | R_ADC_D_SnoozeModePrepare (adc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ADC_D_SnoozeModeExit (adc_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for ADC_D version of the ADC12 peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The ADC module supports the following features:

- 8-, 10-, or 12-bit maximum resolution depending on the MCU
- Configure scans to include:
  - Single channel or Scan channels
  - Temperature sensor channel
  - Voltage sensor channel
  - CTSU TSCAP voltage channel (Not available on all MCUs)
- Configurable scan start trigger:
  - Software scan triggers with no-wait mode or wait mode

    - Hardware scan triggers with no-wait mode or wait mode (timer expiration, for example)
- Configurable scan mode:
    - One-shot scan mode, where each trigger starts a single scan
    - Sequential scan mode, where all channels are scanned continuously
- Optional callback when scan completes
    - Generated an interrupt request when ADLL <= ADCRn <= ADUL
    - Generated an interrupt request when ADUL < ADCRn or ADLL > ADCRn
- Test mode support
    - Checking whether the ADC_D converter is operating normally

# Configuration

### Build Time Configurations for r_adc_d

The following build time configurations are defined in fsp_cfg/r_adc_d_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Internal Reference Voltage Support | • Enabled<br>• Disabled | Disabled | Enable Internal Reference Voltage support for the ADC module. |
| Snooze Mode Support | • Enabled<br>• Disabled | Disabled | Enable Snooze Mode Support. |
| Interrupt Support | • Enabled<br>• Disabled | Enabled | Enable Scan End Interrupt support for the ADC module. |

### Configurations for Analog > ADC (r_adc_d)

This module can be added to the Stacks tab via New Stack > Analog > ADC (r_adc_d).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_adc0 | Module name |
| Resolution | • 12-Bit<br>• 10-Bit<br>• 8-Bit | 12-Bit | Specifies the conversion resolution for this unit. |
| Conversion operation | • One-shot<br>• Sequential | One-shot | Specifies the conversion operation mode. |
| Operation trigger | • Wait<br>• No-wait | Wait | Specifies the operation trigger mode. |

| | | | |
|---|---|---|---|
| Operation voltage | • Normal 1<br>• Normal 2<br>• Low voltage 1<br>• Low voltage 2 | Normal 1 | Specifies operation voltage mode. |
| Conversion Clock (fAD) | • PCLK<br>• PCLK/2<br>• PCLK/4<br>• PCLK/8<br>• PCLK/16<br>• PCLK/32 | PCLK | Specifies divider for conversion clock (fAD). |
| Input | | | |
| Channel Selection Mode | • Select<br>• Scan | Select | Specifies the channel selection mode. |
| A/D Input channel | MCU Specific Options | | Specifies the input channels. |
| Negative Side Reference Voltage | • VSS<br>• VREFL0 | VSS | Selection of the Negative Side Reference Voltage. |
| Positive Side Reference Voltage | • VCC<br>• VREFH0<br>• Internal Reference Voltage | VCC | Selection of the Positive Side Reference Voltage. |
| Interrupts | | | |
| Interrupts > Conversion Result upper/lower bound value setting | | | |
| Generates an interrupt request (INTAD) | • The interrupt signal is output when the ADLL register <= the ADCRn register <= the ADUL register<br>• The interrupt signal is output when the ADCRn register < the ADLL register or the ADUL register < the ADCRn register | The interrupt signal is output when the ADLL register <= the ADCRn register <= the ADUL register | Specify condition generates an interrupt(INTAD) after each time the ADC scan completes. |
| Upper bound (ADUL) value | Must be a valid integer | 255 | Specify the upper limit conversion value that corresponds to the condition to generate an interrupt request (INTAD). |
| Lower bound (ADLL) | Must be a valid integer | 0 | Specify the lower limit |

| | | | |
|---|---|---|---|
| value | | | conversion value that corresponds to the condition to generate an interrupt request (INTAD). |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes. |
| Scan End Interrupt Priority | MCU Specific Options | | Select scan end interrupt priority. |
| Start trigger source | MCU Specific Options | | Specifies the trigger type to be used for this unit. |

### Clock Configuration

The ADC_D conversion clock source may be configured to use ICLK with a selectable division ratio.

The ADC_D clock must be at least 1 MHz when the ADC is used.

### Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC_D.

## Usage Notes

### ADC_D Operational Conversion

The driver supports two operation conversion mode: One-shot scan and Sequential scan modes. For each time conversion, ADC peripheral only converts for each a pin or group of pins depending on channels selection mode, it can be changed to another pin or group of other pins by calling R_ADC_D_ScanCfg().

### Oneshot-scan Mode

In One-shot scan mode, one or group specified channels are scanned once per trigger.

### Sequential-scan Mode

In Sequential scan mode, a single trigger is required to start the scan. Scans continue until R_ADC_D_ScanStop() is called.

### When Interrupt Is Not Enabled

If interrupts are not enabled, the R_ADC_D_StatusGet API can be used to poll the ADC to determine when the scan has completed. The read API function is used to access the converted ADC result.

### When Interrupt Is Enabled

An interrupt is generated depend on setting condition of converted ADC result is in range or out of range of Upper Bound Value (ADUL) and Low Bound Value (ADLL). Refer to hardware manual: ADC12 > Registers to Control the A/D Converte > Figure "ADRCK Bit Interrupt Signal Generation Range".

### Selecting Reference Voltage

The ADC_D positive and negative side reference voltages may be configured for selected MCU's. Please refer to the RA Configuration editor in e2 studio for further details.

*Note*

*When the internal reference voltage is selected as the positive side reference voltage. Please refer condition as below*

- *Select operation voltage is low-voltage mode 1 or 2.*
- *The conversion clock (fAD) must be range 1 to 2 MHZ.*
- *Do not setting select internal reference voltage or temperature sensor channel as an A/D conversion channel.*

### Selecting Internal Reference Voltage or Temperature Sensor channel

When the internal reference voltage or temperature sensor output voltage is selected as the target for A/D conversion, please setting operation voltage is normal mode 2 or low-voltage mode 2.

*Note*

*when using operation voltage is low-voltage mode 2, setting a conversion clock (fAD) is less than or equal 16 MHz.*

### Selecting Conversion Clock

Range for frequency of Conversion Clock (fAD) depends on frequency of ICK. Please refer table as below.

| Frequency of ICLK | Frequency of Conversion Clock (fAD) |
|---|---|
| 4 MHz < ICLK <= 32 MHz | ICLK to ICLK/32 |
| 1 MHz <= fCLK <= 4 MHz | ICLK to ICLK/4 |

### Selecting Operation Voltage

The operation voltage is selectable depending on the analog input channel, VREFH0 voltage, operation mode, and ICLK. For detail, refer to hardware manual: ADC12 > Registers to Control the A/D Converter > tables "A/D conversion time in Normal mode 1 and 2" and "A/D conversion time in Low voltage mode 1 and 2".

*Note*

*When operation voltage is low-voltage modes 1 or 2, setting frequency of conversion clock (fAD) should be used with a frequency less than or equal 24 MHz. See in hardware manual: Electrical Characteristics > Analog Characteristics > table "A/D conversion characteristics in Low-voltage modes 1 and 2".*

### ADC_D conversion in Test Mode

The conversion target for testing can be selected by using the **Input > A/D Input channel** property in the module configuration.

*Note*

*For more details on the method of checking, refer to hardware manual: ADC12 > Testing of the A/D .*

### Using the Temperature Sensor with the ADC_D

The ADC_D HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula, T = (Vs - V1)/slope + T1, where:

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- T2: Temperature at the experimental measurement of another point (degrees C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/degrees C); slope = (V2 - V1)/(T2 - T1)

*Note*

*The slope value can be obtained for each in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.*

For the setting flow, see in hardware manual: ADC12 > Example of Using the ADC12 when Selecting the Temperature Sensor Output Voltage or Internal Reference Voltage, and Software Trigger No-wait Mode and One-shot Conversion Mode.

# Examples

### Basic Example

This is a basic example of minimal use of the ADC_D in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_d_channel_cfg_t elements can be defined
and passed to R_ADC_D_ScanCfg. */
adc_d_channel_cfg_t g_adc_d0_channel_cfg =
{
    .channel_input = ADC_CHANNEL_1,
};
void adc_d_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
}
void adc_d_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
```

```
 /* Initializes the module. */
    err = R_ADC_D_Open(&g_adc_d0_ctrl, &g_adc_d0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
    err = R_ADC_D_ScanCfg(&g_adc_d0_ctrl, &g_adc_d0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_ADC_D_ScanStart(). In other
modes, enable hardware
  * triggers by calling R_ADC_D_ScanStart(). */
    (void) R_ADC_D_ScanStart(&g_adc_d0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_D_StatusGet(&g_adc_d0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t channel1_conversion_result;
    err = R_ADC_D_Read(&g_adc_d0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
    assert(FSP_SUCCESS == err);
}
```

### Temperature Sensor Example

This example shows how to calculate the MCU temperature using the ADC_D and the temperature sensor.

```
#define ADC_D_EXAMPLE_TEMPERATURE_VOLTAGE_V1 (1050000)
#define ADC_D_EXAMPLE_VCC_MICROVOLT (3300000)
#define ADC_D_EXAMPLE_TEMPERATURE_RESOLUTION (4096)
#define ADC_D_EXAMPLE_ADC_TEST_TEMPERATURE_CELSIUS_T1 (25)
#define ADC_PRV_COEFFICIENT (0.5)
void adc_d_temperature_example (void)
```

```
{
 /* The following example calculates the temperature using the data provided.
  * See in hardware manual: Temperature Sensor (TSN) > Preparation for Using the
Temperature Sensor */
 fsp_err_t err = FSP_SUCCESS;
 /* Using normal mode 2 when configure temperature channel */
    g_adc_d0_cfg_extend.operation_voltage = ADC_D_VOLTAGE_MODE_NORMAL_2;
    g_adc_d0_cfg_extend.operation_trigger = ADC_D_TRIGGER_MODE_NO_WAIT;
 /* Initializes the module. */
    err = R_ADC_D_Open(&g_adc_d0_ctrl, &g_adc_d0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure temperature channel */
    g_adc_d0_channel_cfg.channel_input = ADC_CHANNEL_TEMPERATURE;
 /* Enable temperature sensor. */
    err = R_ADC_D_ScanCfg(&g_adc_d0_ctrl, &g_adc_d0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Start of A/D conversion */
    (void) R_ADC_D_ScanStart(&g_adc_d0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_D_StatusGet(&g_adc_d0_ctrl, &status);
    }
 /* The 1st conversion result cannot be used. See in hardware manual: ADC12 > A/D
Converter Setup Procedure >
  * Table "Setup when temperature sensor output voltage and internal reference
voltage is selected" */
 /* Start of A/D conversion */
    (void) R_ADC_D_ScanStart(&g_adc_d0_ctrl);
 /* Wait for conversion to complete. */
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
```

```
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_D_StatusGet(&g_adc_d0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t temperature_conversion_result;
    err = R_ADC_D_Read(&g_adc_d0_ctrl, ADC_CHANNEL_TEMPERATURE,
&temperature_conversion_result);
    assert(FSP_SUCCESS == err);
 /* If the MCU does not provide calibration data, use the value in the hardware
manual or determine it
  * experimentally. */
 /* Get Calibration data from the MCU if available. */
 adc_info_t adc_info;
    (void) R_ADC_D_InfoGet(&g_adc_d0_ctrl, &adc_info);
 /* NOTE: The slope of the temperature sensor varies from sensor to sensor. Renesas
recommends calculating
  * the slope of the temperature sensor experimentally.
  *
  * This example uses the typical slope provided in Table "TSN characteristics" in
the user manual */
    int32_t slope_uv_per_c = BSP_FEATURE_TSN_SLOPE;
 /* Formula for calculating temperature copied from section "Temperature Sensor
(TSN)"
  * of the user manual:
  *
  * Using this, the measured temperature can be calculated according to the following
formula.
  *
  * T = (Vs - V1) / Slope + T1 [C]
  * T: Measured temperature (C)
  * Vs: Voltage output by the temperature sensor when the temperature is measured (V)
  * V1: Voltage output by T1(25 C) is 1.05 V. Refer table "TSN characteristics" in
the user manual
```

```
  * Slope: Temperature slope given in table "TSN characteristics" -3.3 mV/C
  * T1: Temperature of T1 is 25 C.Refer table "TSN characteristics" in the user
manual
  */
    int32_t v1_uv = ADC_D_EXAMPLE_TEMPERATURE_VOLTAGE_V1;
 /* Refer to chapter "Input Voltage and Conversion Results" in the user manual */
    int32_t vs_uv =
       (int32_t) (((temperature_conversion_result - ADC_PRV_COEFFICIENT) *
ADC_D_EXAMPLE_VCC_MICROVOLT) /
                   ADC_D_EXAMPLE_TEMPERATURE_RESOLUTION);
    int32_t temperature_c =
       (int32_t) ((vs_uv - v1_uv) / slope_uv_per_c +
ADC_D_EXAMPLE_ADC_TEST_TEMPERATURE_CELSIUS_T1);
 /* Expect room temperature, break if temperature is outside the range of 20 C to 25
C. */
 if ((temperature_c < 20) || (temperature_c > 25))
    {
       __BKPT(0);
    }
}
```

### Requesting Snooze mode using ADC_D Hardware Trigger

ADC_D can request snooze mode via a hardware trigger and perform an ADC conversion in the following modes:

- Hardware trigger wait mode (select mode, one-shot conversion mode)
- Hardware trigger wait mode (scan mode, one-shot conversion mode)

In order to enable this functionality, the application must call R_ADC_D_SnoozeModePrepare() immediately before entering software standby mode, and call R_ADC_D_SnoozeModeExit() immediately after returning from software standby mode.

```
void adc_d_snooze_request_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Configure ADC_D with the following settings:
  * - adc_d_extended_cfg_t::trigger_source = [ADC_D_TRIGGER_SOURCE_TAU0_TMI01,
```

```
  * ADC_D_TRIGGER_SOURCE_RTC_ALARM_OR_PERIOD,
  * ADC_D_TRIGGER_SOURCE_TML0_ITL0,
  * ADC_D_TRIGGER_SOURCE_ELC] (Hardware trigger)
  * - adc_d_extended_cfg_t::operation_trigger = ADC_D_TRIGGER_MODE_WAIT (Wait mode)
  * - adc_d_extended_cfg_t::conversion_operation = ADC_D_CONVERSION_MODE_ONESHOT (One
shot conversion mode)
  * - ICLK must be operating off of HOCO or MOCO
  */
    err = R_ADC_D_Open(&g_adc_d0_ctrl, &g_adc_d0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_D_ScanCfg(&g_adc_d0_ctrl, &g_adc_d0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure a hardware trigger for ADC_D (Ie. RTC Periodic IRQ). */
    err = R_RTC_C_Open(&g_rtc_ctrl, &g_rtc_cfg);
    assert(FSP_SUCCESS == err);
    err = R_RTC_C_CalendarTimeSet(&g_rtc_ctrl, &g_rtc_current_time);
    assert(FSP_SUCCESS == err);
    err = R_RTC_C_PeriodicIrqRateSet(&g_rtc_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
    assert(FSP_SUCCESS == err);
 /* Configure LPM in snooze mode. */
    err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_snooze_cfg);
    assert(FSP_SUCCESS == err);
 /* Prepare ADC_D to operate in snooze mode. This must be called immediately before
entering software standby mode. */
    err = R_ADC_D_SnoozeModePrepare(&g_adc_d0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Enter low power mode:
  *
  * When RTC_C Periodic IRQ occurs, ICLK source will be started. ADC_D will wait for
the oscillator to stabilize,
  * then start an ADC conversion.
  */
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
    assert(FSP_SUCCESS == err);
```

```
/* After exiting snooze mode, return ADC operation to normal mode. */
    err = R_ADC_D_SnoozeModeExit(&g_adc_d0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | adc_d_channel_cfg_t |
| struct | adc_d_extended_cfg_t |
| struct | adc_d_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | adc_d_channel_mode_t |
| enum | adc_d_voltage_mode_t |
| enum | adc_d_clock_div_t |
| enum | adc_d_trigger_source_t |
| enum | adc_d_trigger_mode_t |
| enum | adc_d_conversion_mode_t |
| enum | adc_d_boundary_t |
| enum | adc_d_negative_vref_t |
| enum | adc_d_positive_vref_t |

## Data Structure Documentation

### ◆ adc_d_channel_cfg_t

| struct adc_d_channel_cfg_t |
|---|
| ADC_D channel(s) configuration |

### ◆ adc_d_extended_cfg_t

| struct adc_d_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for ADC. | | |
| Data Fields | | |
| adc_d_channel_mode_t | channel_mode | ADC_D channels mode setting. |

| | | |
|---|---|---|
| adc_d_voltage_mode_t | operation_voltage | Voltage mode setting. |
| adc_d_clock_div_t | conversion_clockdiv | Divider for conversion clock (fAD) setting. |
| adc_d_trigger_source_t | trigger_source | Trigger source hardware and software setting. |
| adc_d_trigger_mode_t | operation_trigger | Operation mode wait/no wait setting. |
| adc_d_conversion_mode_t | conversion_operation | Sequential/one-shot conversion setting. |
| adc_d_boundary_t | upper_lower_bound | Upper limit and lower limit conversion setting. |
| adc_d_negative_vref_t | negative_vref | Negative side reference voltage setting. |
| adc_d_positive_vref_t | positive_vref | Positive side reference voltage setting. |
| uint8_t | upper_bound_limit | Setting upper limit conversion value. |
| uint8_t | lower_bound_limit | Setting lower limit conversion value. |

### ◆ adc_d_instance_ctrl_t

| struct adc_d_instance_ctrl_t |
|---|
| ADC_D instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). |

## Enumeration Type Documentation

### ◆ adc_d_channel_mode_t

| enum adc_d_channel_mode_t | |
|---|---|
| ADC_D channels mode selection | |
| Enumerator | |
| ADC_D_CHANNEL_MODE_SELECT | Select mode. |
| ADC_D_CHANNEL_MODE_SCAN | Scan mode. |

### ◆ adc_d_voltage_mode_t

| enum adc_d_voltage_mode_t | |
|---|---|
| Operation voltage mode selection | |
| Enumerator | |
| ADC_D_VOLTAGE_MODE_NORMAL_1 | Normal 1. |
| ADC_D_VOLTAGE_MODE_NORMAL_2 | Normal 2. |
| ADC_D_VOLTAGE_MODE_LOW_1 | Low voltage 1. |
| ADC_D_VOLTAGE_MODE_LOW_2 | Low voltage 2. |

### ◆ adc_d_clock_div_t

| enum adc_d_clock_div_t | |
|---|---|
| Divider for Conversion Clock (fAD) | |
| Enumerator | |
| ADC_D_CLOCK_DIV_32 | ADC_D clock division ICLK/32. |
| ADC_D_CLOCK_DIV_16 | ADC_D clock division ICLK/16. |
| ADC_D_CLOCK_DIV_8 | ADC_D clock division ICLK/8. |
| ADC_D_CLOCK_DIV_4 | ADC_D clock division ICLK/4. |
| ADC_D_CLOCK_DIV_2 | ADC_D clock division ICLK/2. |
| ADC_D_CLOCK_DIV_1 | ADC_D clock division ICLK/1. |

### ◆ adc_d_trigger_source_t

| enum adc_d_trigger_source_t | |
|---|---|
| Selection trigger signal | |
| Enumerator | |
| ADC_D_TRIGGER_SOURCE_TAU0_TMI01 | Timer channel 01 count or capture end interrupt signal. |
| ADC_D_TRIGGER_SOURCE_RTC_ALARM_OR_PERIOD | Realtime clock interrupt signal. |
| ADC_D_TRIGGER_SOURCE_TML0_ITL0 | 32-bit interval timer interrupt signal |
| ADC_D_TRIGGER_SOURCE_ELC | Event input from ELC. |
| ADC_D_TRIGGER_SOURCE_SOFTWARE | Software trigger, this option is controlled by bit ADCS, ADCE. |

### ◆ adc_d_trigger_mode_t

| enum adc_d_trigger_mode_t | |
|---|---|
| Select trigger mode | |
| Enumerator | |
| ADC_D_TRIGGER_MODE_NO_WAIT | Trigger no-wait mode. |
| ADC_D_TRIGGER_MODE_WAIT | Trigger wait mode. |

### ◆ adc_d_conversion_mode_t

| enum adc_d_conversion_mode_t | |
|---|---|
| Select conversion operation mode | |
| Enumerator | |
| ADC_D_CONVERSION_MODE_SEQUENTIAL | Continuous conversion mode. |
| ADC_D_CONVERSION_MODE_ONESHOT | Single conversion mode. |

### ◆ adc_d_boundary_t

| enum adc_d_boundary_t | |
|---|---|
| Select the upper limit and lower limit conversion result values | |
| Enumerator | |
| ADC_D_BOUNDARY_IN_RANGE | The interrupt signal (INTAD) is output in range ADLL and AULL. |
| ADC_D_BOUNDARY_OUT_OF_RANGE | The interrupt signal (INTAD) is output out of range ADLL and AULL. |

### ◆ adc_d_negative_vref_t

| enum adc_d_negative_vref_t | |
|---|---|
| The negative side reference voltage selection | |
| Enumerator | |
| ADC_D_NEGATIVE_VREF_VSS | Supplied from VSS. |
| ADC_D_NEGATIVE_VREF_VREFL0 | Supplied from VREFL0. |

### ◆ adc_d_positive_vref_t

| enum adc_d_positive_vref_t | |
|---|---|
| The positive side reference voltage selection | |
| Enumerator | |
| ADC_D_POSITIVE_VREF_VCC | Supplied from VSS. |
| ADC_D_POSITIVE_VREF_VREFH0 | Supplied from VREFH0. |
| ADC_D_POSITIVE_VREF_IVREF | Supplied from the internal reference voltage. |

## Function Documentation

### ◆ R_ADC_D_Open()

fsp_err_t R_ADC_D_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Initialize the ADC_D peripheral. If interrupt is enabled, the function registers a callback function for notifying the user when a scan has completed. Implements adc_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is ready for use. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |
| FSP_ERR_IRQ_BSP_DISABLED | A callback is provided, but the interrupt is not enabled. |
| FSP_ERR_INVALID_HW_CONDITION | Invalid configuration corresponds to condition HardWare UM. |

### ◆ R_ADC_D_ScanCfg()

fsp_err_t R_ADC_D_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_channel_cfg* )

Configures the ADC_D scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_d_channel_cfg_t to p_channel_cfg.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel specific settings applied. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_INVALID_HW_CONDITION | Invalid configuration corresponds to condition HardWare UM. |
| FSP_ERR_INVALID_STATE | Invalid Scan Configuration. |
| FSP_ERR_INVALID_CHANNEL | Channel is invalid. |

### ◆ R_ADC_D_CallbackSet()

fsp_err_t R_ADC_D_CallbackSet ( adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |

### ◆ R_ADC_D_ScanStart()

fsp_err_t R_ADC_D_ScanStart ( adc_ctrl_t * *p_ctrl*)

Starts a software trigger scan or enables the hardware trigger mode for a scan depending on how the triggers were configured in the R_ADC_D_Open call. If the unit was configured for ELC or interrupt hardware triggering, then this function allows the trigger signal to get to the ADC_D. The function is not able to control the generation of the trigger itself.

**Precondition**

Call R_ADC_D_ScanCfg after R_ADC_D_Open before starting a scan.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started (software trigger) or hardware trigger mode enabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |
| FSP_ERR_NOT_INITIALIZED | ADC_D is not initialized. |

### ◆ R_ADC_D_ScanGroupStart()

fsp_err_t R_ADC_D_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id* )

adc_api_t::scanStart is not supported on the ADC_D. Use scanStart instead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_ADC_D_ScanStop()

fsp_err_t R_ADC_D_ScanStop ( adc_ctrl_t * *p_ctrl*)

Disables the hardware trigger for a scan or select mode and immediately stops converters. This function will abort conversions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped (software trigger) or hardware triggers disabled. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |

### ◆ R_ADC_D_StatusGet()

fsp_err_t R_ADC_D_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Provides the status of any scan process that was started, including scans started by ELC or interrupts triggers.

***Note***

*In Hardware no-wait mode, ADCS retains the value 1 after conversion end.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module status stored in the provided pointer p_status. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |

### ◆ R_ADC_D_Read()

fsp_err_t R_ADC_D_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads conversion results from a channel or sensor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |
| FSP_ERR_NOT_INITIALIZED | Unit is not initialized. |
| FSP_ERR_INVALID_MODE | Invalid configuration for channel_mode. |

### ◆ R_ADC_D_Read32()

fsp_err_t R_ADC_D_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads conversion results from a select/scan channel or sensor register into a 32-bit result.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read into provided p_data. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |
| FSP_ERR_NOT_INITIALIZED | ADC_D is not initialized. |
| FSP_ERR_INVALID_MODE | Invalid configuration for channel_mode. |

### ◆ R_ADC_D_InfoGet()

fsp_err_t R_ADC_D_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

Get information of address ADCR to reading the data, determine the size of data that must be read, size data of each transfer, name of the ELC event for the peripheral, name of the peripheral in the ELC list

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |
| FSP_ERR_NOT_INITIALIZED | ADC_D is not initialized. |

### ◆ R_ADC_D_Close()

fsp_err_t R_ADC_D_Close ( adc_ctrl_t * *p_ctrl*)

This function ends any scan or select mode in progress, disables interrupts, and removes power to the A/D peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |

### ◆ R_ADC_D_Calibrate()

fsp_err_t R_ADC_D_Calibrate ( adc_ctrl_t *const *p_ctrl*, void const * *p_extend* )

adc_api_t::calibrate is not supported on the ADC_D.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_ADC_D_OffsetSet()

fsp_err_t R_ADC_D_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset* )

adc_api_t::offsetSet is not supported on the ADC_D.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_ADC_D_SnoozeModePrepare()

fsp_err_t R_ADC_D_SnoozeModePrepare ( adc_ctrl_t *const *p_ctrl*)

Prepare ADC_D to enter snooze mode via a hardware trigger. This function must be called immediately before entering software standby mode in order to allow the configured hardware trigger to transition the MCU from software standby mode to snooze mode and perform an ADC conversion.

Supported modes for requesting snooze mode via hardware trigger:

- channel_mode = ADC_D_CHANNEL_MODE_SELECT, conversion_operation = ADC_D_CONVERSION_MODE_ONESHOT
- channel_mode = ADC_D_CHANNEL_MODE_SCAN, conversion_operation = ADC_D_CONVERSION_MODE_ONESHOT

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the ADC control block |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ADC is configured to request Snooze mode. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | ADC_D is not open. |
| FSP_ERR_INVALID_MODE | ADC is in an invalid mode for requesting Snooze mode. |

### ◆ R_ADC_D_SnoozeModeExit()

| fsp_err_t R_ADC_D_SnoozeModeExit ( adc_ctrl_t *const *p_ctrl*) |
|---|
| After exiting snooze mode, if the ADC_D module was in snooze mode, then this function must be called in order to restore ADC operation to normal mode. |

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the ADC control block |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ADC is configured to request Snooze mode. |
| FSP_ERR_INVALID_MODE | ADC is in an invalid mode for requesting Snooze mode. |

## 5.2.1.4 Comparator, High-Speed (r_acmphs)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_ACMPHS_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| fsp_err_t | R_ACMPHS_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| fsp_err_t | R_ACMPHS_OutputEnable (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPHS_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| fsp_err_t | R_ACMPHS_Close (comparator_ctrl_t *p_ctrl) |

### Detailed Description

Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface.

# Overview

### Features

The ACMPHS HAL module supports the following features:

- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option for comparator output on VCOUT, CMPOUTn[1], or CMPOUT012[1] pin
- ELC event output

*Note*

*1. This output pin is not available on all MCUs.*

# Configuration

### Build Time Configurations for r_acmphs

The following build time configurations are defined in fsp_cfg/r_acmphs_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > Comparator, High-Speed (r_acmphs)

This module can be added to the Stacks tab via New Stack > Analog > Comparator, High-Speed (r_acmphs).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comparator0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the hardware channel. |
| Trigger Edge Selector | • Rising<br>• Falling<br>• Both Edge | Both Edge | The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL. |
| Noise Filter | • No Filter<br>• 8<br>• 16<br>• 32 | No Filter | Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update. |
| Maximum status retries (CMPMON) | Must be a valid non-negative integer between 2 and 32-bit maximum value | 1024 | Maximum number of status retries. |

| | | | |
|---|---|---|---|
| Output Polarity | • Not Inverted<br>• Inverted | Not Inverted | When enabled comparator output is inverted. This affects the output read from R_ACMPHS_StatusGet() , the pin output level, and the edge trigger. |
| Pin Output | • Disabled<br>• Enabled | Disabled | Turn this on to enable the CMPOUTn signal for this channel. The CMPOUTn signal for each channel is OR'd together and the result is output to VCOUT. More pin output options are available on select MCUs. |
| Callback | Name must be a valid C symbol | NULL | Define this function in the application. It is called when the Trigger event occurs. |
| Comparator Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the comparator interrupt. |
| Analog Input Voltage Source (IVCMP) | MCU Specific Options | | Select the Analog input voltage source. Channel mentioned in the options represents channel in ACMPHS |
| Reference Voltage Input Source (IVREF) | MCU Specific Options | | Select the Analog reference voltage source. Channel mentioned in the options represents channel in ACMPHS |

**Clock Configuration**

The ACMPHS peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

**Pin Configuration**

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The IVCMPn pins are used as comparator inputs. The IVREFn pins are used as comparator reference values.

# Usage Notes

### Noise Filter

When the noise filter is enabled, the ACMPHP0/ACMPHP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

### Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPHS_StatusGet() will return an inverted status.

### Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.
- When the noise filter is not enabled the hardware requires software debouncing of the output (two consecutive equal values). This is automatically managed in R_ACMPHS_StatusGet but may result in delay or an API error in rare edge cases.
- Constraints apply on the simultaneous use of ACMPHS analog input and ADC analog input. Refer to the "Usage Notes" section in your MCU's User's Manual for the ADC unit(s) for more details.
- To allow ACMPHS0 to cancel Software Standby mode or enter Snooze, set the CSTEN bit to 1 and the CDFS bits to 00 in the CMPCTL0 register.

## Examples

### Basic Example

The following is a basic example of using the ACMPHS to detect when the analog voltage input to IVCMP rises above the analog voltage input to IVREF. A GPIO output acts as the comparator input and is externally connected to the IVCMP input of the ACMPHS. An analog voltage input should also be supplied to the IVREF input pin.

```
#define ADC_PGA_BYPASS_VALUE (0x9999)
/* Connect this control pin to the IVCMP input of the comparator. This can be any
GPIO pin
 * that is not input only. */
#define ACMPHS_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_05_PIN_03)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmphs_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmphs_example ()
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /*
  * Start with the IVCMP pin low. This example assumes the comparator is configured
to trigger
  * when the voltage of the analog input to IVCMP rises above voltage of the analog
input to
  * IVREF.
  */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPHS module */
    err = R_ACMPHS_Open(&g_comparator_ctrl, &g_comparator_cfg);
    assert(FSP_SUCCESS == err);
 /*
  * If an ADC PGA exists for the analog input pin, then the PGA must be manually
configured in order for the pin to be used as
  * an IVCMP input. This procedure is slightly different depending on the MCU (See
below).
  */
#if BSP_MCU_GROUP_RA6M3
 /* The following applies for MCUs with the ADC peripheral:
  *
  * Bypass the PGA on ADC unit 0.
  * (See table "Input source configuration of the ACMPHS" in the relevant hardware
manual) */
 R_BSP_MODULE_START(FSP_IP_ADC, 0);
    R_ADC0->ADPGACR   = ADC_PGA_BYPASS_VALUE;
    R_ADC0->ADPGADCR0 = 0;
#elif BSP_MCU_GROUP_RA6T2
 /* The following applies for MCUs with the ADC_B peripheral:
  *
  * Configure PGA on ADC unit 0.
```

```
  * (See table "PGA Settings and Available Related Functions" in the A/D Converter
Operation section of the relevant
  * hardware manual) */
 R_BSP_MODULE_START(FSP_IP_ADC, 0);
    R_ADC_B->ADPGACR[0] = R_ADC_B0_ADPGACR_PGAGEN_Msk;
#endif
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPHS_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable the comparator output */
    (void) R_ACMPHS_OutputEnable(&g_comparator_ctrl);
 /* Set the IVCMP pin high. */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
 while (0 == g_comparator_events)
    {
 /* Wait for interrupt. */
    }
 comparator_status_t status;
 /* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
    (void) R_ACMPHS_StatusGet(&g_comparator_ctrl, &status);
}
```

### Function Documentation

### ◆ R_ACMPHS_Open()

fsp_err_t R_ACMPHS_Open ( comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg* )

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL |
| FSP_ERR_INVALID_ARGUMENT | An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation. |
| FSP_ERR_ALREADY_OPEN | The control block is already open or the hardware lock is taken. |

### ◆ R_ACMPHS_InfoGet()

fsp_err_t R_ACMPHS_InfoGet ( comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_OutputEnable()

fsp_err_t R_ACMPHS_OutputEnable ( comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Comparator output is enabled. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_StatusGet()

fsp_err_t R_ACMPHS_StatusGet ( comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status* )

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of the comparator is provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_TIMEOUT | The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts. |

### ◆ R_ACMPHS_Close()

fsp_err_t R_ACMPHS_Close ( comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### 5.2.1.5 Comparator, High-Speed (r_acmphs_b)

Modules » Analog

**Functions**

| | |
|---|---|
| fsp_err_t | R_ACMPHS_B_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| fsp_err_t | R_ACMPHS_B_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| fsp_err_t | R_ACMPHS_B_OutputEnable (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPHS_B_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| fsp_err_t | R_ACMPHS_B_Close (comparator_ctrl_t *p_ctrl) |

**Detailed Description**

Driver for the ACMPHS_B peripheral on RA MCUs. This module implements the Comparator Interface.

# Overview

**Features**

The ACMPHS HAL module supports the following features:

- Callback on rising edge, falling edge or both
- Two independent comparators (Channel 0,Channel 1)
- Configurable debounce filter
- Optional output (VCOUT) for each comparator
- ELC event output for each comparator

# Configuration

**Build Time Configurations for r_acmphs_b**

The following build time configurations are defined in fsp_cfg/r_acmphs_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Analog > Comparator, High-Speed (r_acmphs_b)**

This module can be added to the Stacks tab via New Stack > Analog > Comparator, High-Speed (r_acmphs_b).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comparator0 | Module name. |
| Channel | MCU Specific Options | | Select the hardware channel. |
| Trigger Edge Selector | • Rising<br>• Falling<br>• Both Edge | Both Edge | The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL. |
| Noise Filter | • No sampling (bypass)<br>• Sampling at PCLKB<br>• Sampling at PCLKB/8 | No sampling (bypass) | Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update. |
| Output Polarity | • Not Inverted<br>• Inverted | Not Inverted | When enabled comparator output is inverted. This affects the output read from R_ACMPHS_B_StatusGet(), the pin output level, and the edge trigger. |
| Pin Output | • Disabled<br>• Enabled | Disabled | Turn this on to enable the CMPOUTn signal for this channel. The CMPOUTn signal for each channel is OR'd together and the result is output to VCOUT. More pin output options are available on select MCUs. |
| Callback | Name must be a valid C symbol | NULL | Define this function in the application. It is called when the Trigger event occurs. |
| Comparator Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the comparator interrupt. |
| Analog Input Voltage Source | MCU Specific Options | | Select the Analog input voltage source. Channel mentioned in |

| | | |
|---|---|---|
| | | the options represents channel in ACMPHS_B |
| Reference Voltage Input Source | MCU Specific Options | Select the Analog reference voltage source. Channel mentioned in the options represents channel in ACMPHS_B |

### Clock Configuration

The ACMPHS peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

## Usage Notes

### Noise Filter

When the noise filter is enabled for a channel, the comparison is evaluated three times before the output of the comparison in VCOUT or ELC event is affected. The sampling rate is driven by PCLKB. The sampling can be further slowed by adjusting the comparator_filter_t setting. NOTE: The R_ACMPLP_StatusGet reflects the comparison results before the effects of the noise filter.

### Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPHS_StatusGet() will return an inverted status.

### Limitations

Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.

## Examples

### Basic Example

The following is a basic example of minimal use of the ACMPHS_B. The comparator is configured to trigger a callback when the input rises above the internal reference voltage (VREF). A GPIO output acts as the comparator input and is externally connected to the CMPIN input of the ACMPHS_B.

```
/* Connect this control pin to the IVCMP input of the comparator. This can be any
GPIO pin
 * that is not input only. */
#define ACMPHS_B_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_01_PIN_00)
```

```
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmphs_b_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmphs_b_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /*
  * Start with the IVCMP pin low. This example assumes the comparator is configured
to trigger
  * when the voltage of the analog input to IVCMP rises above voltage of the internal
Vref.
  */
    (void) R_BSP_PinWrite(ACMPHS_B_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPHS_B module */
    err = R_ACMPHS_B_Open(&g_comparator_ctrl, &g_comparator_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPHS_B_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable the comparator output */
    (void) R_ACMPHS_B_OutputEnable(&g_comparator_ctrl);
 /* Set the IVCMP pin high. */
    (void) R_BSP_PinWrite(ACMPHS_B_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
 while (0 == g_comparator_events)
    {
 /* Wait for interrupt. */
    }
```

```
comparator_status_t status;
/* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
   (void) R_ACMPHS_B_StatusGet(&g_comparator_ctrl, &status);
}
```

## Function Documentation

### ◆ R_ACMPHS_B_Open()

fsp_err_t R_ACMPHS_B_Open ( comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg* )

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL |
| FSP_ERR_INVALID_ARGUMENT | An argument is invalid. Filter of 16 (COMPARATOR_FILTER_16), filter of 32 (COMPARATOR_FILTER_32) are not supported in this implementation. |
| FSP_ERR_ALREADY_OPEN | The control block is already open or the hardware lock is taken. |
| FSP_ERR_IN_USE | The channel is already in use. |
| FSP_ERR_INVALID_CHANNEL | Selected channel does not exist. |
| FSP_ERR_UNSUPPORTED | Selected mode not supported by this API |

### ◆ R_ACMPHS_B_InfoGet()

fsp_err_t R_ACMPHS_B_InfoGet ( comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_B_OutputEnable()

fsp_err_t R_ACMPHS_B_OutputEnable ( comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Comparator output is enabled. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_B_StatusGet()

fsp_err_t R_ACMPHS_B_StatusGet ( comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status* )

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of the comparator is provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPHS_B_Close()

fsp_err_t R_ACMPHS_B_Close ( comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## 5.2.1.6 Comparator, Low-Power (r_acmplp)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_ACMPLP_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| fsp_err_t | R_ACMPLP_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| fsp_err_t | R_ACMPLP_OutputEnable (comparator_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ACMPLP_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| fsp_err_t | R_ACMPLP_Close (comparator_ctrl_t *p_ctrl) |

### Detailed Description

Driver for the ACMPLP peripheral on RA MCUs. This module implements the Comparator Interface.

# Overview

### Features

The ACMPLP HAL module supports the following features:

- Normal mode or window mode
- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option for comparator output on VCOUT pin
- ELC event output

# Configuration

### Build Time Configurations for r_acmplp

The following build time configurations are defined in fsp_cfg/r_acmplp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Reference Voltage Selection for ACMPLP1 (Standard mode only) | • IVREF0<br>• IVREF1 | IVREF1 | ACMPLP1 may optionally be configured to use IVREF0 as a reference input instead of IVREF1. Note that if |

| | | | |
|---|---|---|---|
| | | | IVREF0 is selected, ACMPLP0 and ACMPLP1 must use the same setting for IVREF. |

### Configurations for Analog > Comparator, Low-Power (r_acmplp)

This module can be added to the Stacks tab via New Stack > Analog > Comparator, Low-Power (r_acmplp).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comparator0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the hardware channel. |
| Mode | • Standard<br>• Window | Standard | In standard mode, comparator output is high if VCMP > VREF. In window mode, comparator output is high if VCMP is outside the range of VREF0 to VREF1. |
| Trigger | • Rising<br>• Falling<br>• Both Edge | Both Edge | The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL. |
| Filter | • No sampling (bypass)<br>• Sampling at PCLKB<br>• Sampling at PCLKB/8<br>• Sampling at PCLKB/32 | No sampling (bypass) | Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update. |
| Output Polarity | • Not Inverted<br>• Inverted | Not Inverted | When enabled comparator output is inverted. This affects the output read from R_ACMPLP_StatusGet(), the pin output level, and the edge trigger. |
| Pin Output (VCOUT) | • Disabled<br>• Enabled | Disabled | Turn this on to include the output from this comparator on VCOUT. The comparator output on VCOUT is OR'd with |

| | | | |
|---|---|---|---|
| | | | output from all other ACMPHS and ACMPLP comparators. |
| Vref (Standard mode only) | • Enabled<br>• Disabled | Disabled | If reference voltage selection is enabled then internal reference voltage is used as comparator input |
| Callback | Name must be a valid C symbol | NULL | Define this function in the application. It is called when the Trigger event occurs. |
| Comparator Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the comparator interrupt. |
| Analog Input Voltage Source (IVCMP) | MCU Specific Options | | Select the comparator input source. Only options for the configured channel are valid. |
| Reference Voltage Input Source (IVREF) | MCU Specific Options | | Select the comparator reference voltage source.<br><br>If channel 1 is seleected and the 'Reference Voltage Selection (ACMPLP1)' config option is set to IVREF0, select one of the Channel 0 options. In all other cases, only options for the configured channel are valid. |

**Clock Configuration**

The ACMPLP peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

**Pin Configuration**

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The CMPINn pins are used as comparator inputs. The CMPREFn pins are used as comparator reference values.

# Usage Notes

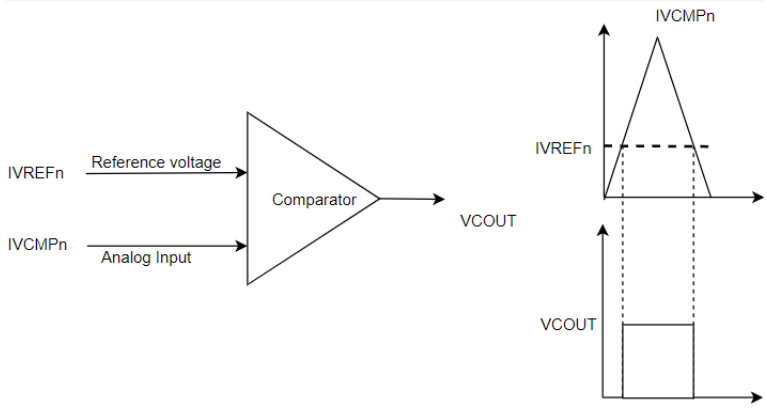



Figure 166: ACMPLP Standard Mode Operation

### Noise Filter

When the noise filter is enabled, the ACMPLP0/ACMPLP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

### Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPLP_StatusGet() will return an inverted status.

### Window Mode

In window mode, the comparator indicates if the analog input voltage falls within the window (low and high reference voltage) or is outside the window.

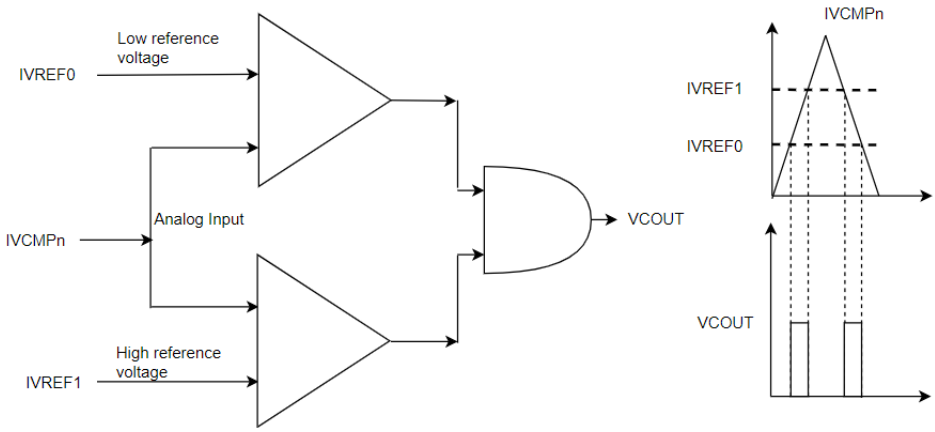



Figure 167: ACMPLP Window Mode Operation

### Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.
- Low speed is not supported by the ACMPLP driver.

# Examples

### Basic Example

The following is a basic example of minimal use of the ACMPLP. The comparator is configured to trigger a callback when the input rises above the internal reference voltage (VREF). A GPIO output acts as the comparator input and is externally connected to the CMPIN input of the ACMPLP.

```
/* Connect this control pin to the VCMP input of the comparator. This can be any GPIO
pin
 * that is not input only. */
#define ACMPLP_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_04_PIN_08)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmplp_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmplp_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /* Start with the VCMP pin low. This example assumes the comparator is configured to
trigger
  * when VCMP rises above VREF. */
    (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPLP module */
    err = R_ACMPLP_Open(&g_comparator_ctrl, &g_comparator_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPLP_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable the comparator output */
```

```
    (void) R_ACMPLP_OutputEnable(&g_comparator_ctrl);
/* Set VCMP low. */
    (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
while (0 == g_comparator_events)
    {
/* Wait for interrupt. */
    }
comparator_status_t status;
/* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
    (void) R_ACMPLP_StatusGet(&g_comparator_ctrl, &status);
}
```

## Enumerations

| | |
|---|---|
| enum | acmplp_input_t |
| enum | acmplp_reference_t |

## Enumeration Type Documentation

### ◆ acmplp_input_t

| enum acmplp_input_t | |
|---|---|
| Enumerator | |
| ACMPLP_INPUT_AMPO | Not available on all MCUs. |
| ACMPLP_INPUT_CMPIN_1 | Not available on all MCUs. |

### ◆ acmplp_reference_t

| enum acmplp_reference_t | |
|---|---|
| Enumerator | |
| ACMPLP_REFERENCE_CMPREF_1 | Not available on all MCUs. |

## Function Documentation

### ◆ R_ACMPLP_Open()

fsp_err_t R_ACMPLP_Open ( comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg* )

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL |
| FSP_ERR_INVALID_ARGUMENT | An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation. p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. |
| FSP_ERR_ALREADY_OPEN | The control block is already open or the hardware lock is taken. |
| FSP_ERR_IN_USE | The channel is already in use. |

### ◆ R_ACMPLP_InfoGet()

fsp_err_t R_ACMPLP_InfoGet ( comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_OutputEnable()

fsp_err_t R_ACMPLP_OutputEnable ( comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Comparator output is enabled. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_StatusGet()

fsp_err_t R_ACMPLP_StatusGet ( comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status* )

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of the comparator is provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_ACMPLP_Close()

fsp_err_t R_ACMPLP_Close ( comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## 5.2.1.7 DAC (r_dac)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_DAC_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg) |
| fsp_err_t | R_DAC_Write (dac_ctrl_t *p_api_ctrl, uint16_t value) |
| fsp_err_t | R_DAC_Start (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_Stop (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_Close (dac_ctrl_t *p_api_ctrl) |

### Detailed Description

Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface.

# Overview

### Features

The DAC module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Supports output amplifiers on selected MCUs
- Supports charge pump on selected MCUs
- Supports synchronization with the Analog-to-Digital Converter (ADC) module

# Configuration

*Note*

*For MCUs supporting more than one channel, the following configuration options are shared by all the DAC channels:*

- *Synchronize with ADC*
- *Data Format*
- *Charge Pump*

### Build Time Configurations for r_dac

The following build time configurations are defined in fsp_cfg/r_dac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > DAC (r_dac)

This module can be added to the Stacks tab via New Stack > Analog > DAC (r_dac). Non-secure

callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_dac0 | Module name. |
| Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the hardware channel. |
| Synchronize with ADC | MCU Specific Options | | Enable DA/AD synchronization. |
| Data Format | • Right Justified<br>• Left Justified | Right Justified | Specify the DAC data format. |
| Output Amplifier | MCU Specific Options | | Enable the DAC output amplifier. |
| Charge Pump (Requires MOCO active) | MCU Specific Options | | Enable the DAC charge pump. |
| Internal Output | MCU Specific Options | | Enable DAC output to internal modules. |
| ELC Trigger Source | MCU Specific Options | | ELC event source that will trigger the DAC to start a conversion. |
| Reference Voltage | MCU Specific Options | | Select the DAC reference voltage. |

### Clock Configuration

The DAC peripheral module uses PCLKB as its clock source.

### Pin Configuration

The DAn pins are used as analog outputs. Each DAC channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply pins for the DAC and ADC.

The VREFH and VREFL pins are top and ground voltage reference pins for the DAC and ADC.

## Usage Notes

### Charge Pump

The charge pump must be enabled when using DAC pin output while operating at $AV_{CC}$ < 2.7V.

*Note*

*The MOCO must be running to use the charge pump.*
*If the DAC output is to be routed to an internal signal, do not enable the charge pump.*

### Output to Internal Modules

The DAC output can be used as an analog input to other peripherals on the MCU (eg. ACMPHS, ADC) without outputting the voltage to an external pin.

On some MCUs this functionality must be enabled during configuration using dac_extended_cfg_t::internal_output_enabled. When internal output is enabled, the DAC output will be routed to internal modules. If the DAC output amplifier is enabled or when internal output is disabled, the output will be routed to the DAC output pin (DAn).

### Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC conversion is started it will automatically be delayed until after the ADC conversion is complete.

### Limitations

- For MCUs supporting ADC unit 1:
  - Once synchronization between DAC and ADC unit 1 is turned on during R_DAC_Open synchronization cannot be turned off by the driver. In order to desynchronize DAC with ADC unit 1, manually clear DAADSCR.DAADST to 0 when the ADCSR.ADST bit is 0 and ADC unit 1 is halted.
  - The DAC module can only be synchronized with ADC unit 1.
  - For MCUs having more than 1 DAC channel, both channels are synchronized with ADC unit 1 if synchronization is enabled.

## Examples

### Basic Example

This is a basic example of minimal use of the R_DAC in an application. This example shows how this driver can be used for basic Digital to Analog Conversion operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA0 as Analog. */
 /* Initialize the DAC channel */
    err = R_DAC_Open(&g_dac_ctrl, &g_dac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    value = (uint16_t) DAC_EXAMPLE_VALUE_ABC;
    err = R_DAC_Write(&g_dac_ctrl, value);
    assert(FSP_SUCCESS == err);
    err = R_DAC_Start(&g_dac_ctrl);
    assert(FSP_SUCCESS == err);
```

```
}
```

### Data Structures

| | |
|---|---|
| struct | dac_instance_ctrl_t |
| struct | dac_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | dac_ref_volt_sel_t |

## Data Structure Documentation

### ◆ dac_instance_ctrl_t

| struct dac_instance_ctrl_t |
|---|
| DAC instance control block. |

### ◆ dac_extended_cfg_t

| struct dac_extended_cfg_t | | |
|---|---|---|
| DAC extended configuration | | |
| Data Fields | | |
| bool | enable_charge_pump | Enable DAC charge pump available on selected MCUs. |
| bool | output_amplifier_enabled | Output amplifier enable available on selected MCUs. |
| bool | internal_output_enabled | Internal output enable available on selected MCUs. |
| dac_data_format_t | data_format | Data format. |
| dac_ref_volt_sel_t | ref_volt_sel | Reference voltage selection. |

## Enumeration Type Documentation

### ◆ dac_ref_volt_sel_t

| enum dac_ref_volt_sel_t | |
|---|---|
| DAC Reference voltage selection. | |
| Enumerator | |
| DAC_VREF_NONE | No reference voltage selected. |
| DAC_VREF_AVCC0_AVSS0 | Select AVCC0/AVSS0. |
| DAC_VREF_IVREF_AVSS0 | Select Internal reference voltage/AVSS0. |
| DAC_VREF_VREFH_VREFL | Select VREFH/VREFL. |

## Function Documentation

### ◆ R_DAC_Open()

fsp_err_t R_DAC_Open ( dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg* )

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. One or both of the following parameters may be NULL: p_api_ctrl or p_cfg<br>2. data_format value in p_cfg is out of range.<br>3. Extended configuration structure is set to NULL for MCU supporting charge pump. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel ID requested in p_cfg may not available on the devices. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |

### ◆ R_DAC_Write()

fsp_err_t R_DAC_Write ( dac_ctrl_t * *p_api_ctrl*, uint16_t *value* )

Write data to the D/A converter and enable the output if it has not been enabled.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Start()

fsp_err_t R_DAC_Start ( dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output if it has not been started.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is started successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_IN_USE | Attempt to re-start a channel. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Stop()

fsp_err_t R_DAC_Stop ( dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The control is successfully stopped. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_Close()

| fsp_err_t R_DAC_Close ( dac_ctrl_t * *p_api_ctrl*) |
|---|
| Stop the D/A conversion, stop output, and close the DAC channel.<br><br>**Return values**<br>FSP_SUCCESS — The channel is successfully closed.<br>FSP_ERR_ASSERTION — p_api_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — Channel associated with p_ctrl has not been opened. |

#### 5.2.1.8 DAC8 (r_dac8)

Modules » Analog

**Functions**

| | |
|---|---|
| fsp_err_t | R_DAC8_Open (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg) |
| fsp_err_t | R_DAC8_Write (dac_ctrl_t *const p_ctrl, uint16_t value) |
| fsp_err_t | R_DAC8_Start (dac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_DAC8_Stop (dac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_DAC8_Close (dac_ctrl_t *const p_ctrl) |

**Detailed Description**

Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface.

# Overview

### Features

The DAC8 module outputs one of 256 voltage levels between the positive and negative reference voltages. DAC8 on selected MCUs have below features

- Charge pump control
- Synchronization with the Analog-to-Digital Converter (ADC) module
- Multiple Operation Modes
  - Normal
  - Real-Time (Event Link)

# Configuration

*Note*

*For MCUs supporting more than one channel, the following configuration options are shared by all the DAC8 channels:*

- *Synchronize with ADC*
- *Charge Pump*

### Build Time Configurations for r_dac8

The following build time configurations are defined in fsp_cfg/r_dac8_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > DAC8 (r_dac8)

This module can be added to the Stacks tab via New Stack > Analog > DAC8 (r_dac8).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_dac8_0 | Module name. |
| Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the hardware channel. |
| D/A A/D Synchronous Conversion | MCU Specific Options | | Synchronize the DAC8 update with the ADC to reduce interference with A/D conversions. |
| DAC Mode | MCU Specific Options | | Select the DAC operating mode |
| Real-time Trigger Event | MCU Specific Options | | Specify the event used to trigger conversion in Real-time mode. This setting is only valid when Real-time mode is enabled. |
| Charge Pump (Requires MOCO active) | MCU Specific Options | | Enable the DAC charge pump. |

### Clock Configuration

The DAC8 peripheral module uses the PCLKB as its clock source.

### Pin Configuration

The DA8_n pins are used as analog outputs. Each DAC8 channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply and reference pins for the DAC8.

# Usage Notes

### Charge Pump

The charge pump must be enabled when using DAC8 pin output while operating at $AV_{CC}$ < 2.7V.

*Note*

*The MOCO must be running to use the charge pump.*
*If DAC8 output is to be routed to an internal signal, do not enable the charge pump.*

### Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC8 conversion is started it will automatically be delayed until after the ADC conversion is complete.

### Real-time Mode

When Real-time mode is selected, the DAC8 will perform a conversion each time the selected ELC event is received.

### Limitations

- Synchronization between DAC8 and ADC is activated when calling R_DAC8_Open. At this point synchronization cannot be deactivated by the driver. In order to desynchronize DAC8 with ADC, manually clear DACADSCR.DACADST to 0 while the ADCSR.ADST bit is 0 and the ADC is halted.
- For MCUs having more than 1 DAC8 channel, both channels are synchronized with ADC if synchronization is enabled.

# Examples

### Basic Example

This is a basic example of minimal use of the R_DAC8 in an application. This example shows how this driver can be used for basic 8 bit Digital to Analog Conversion operations.

```
dac8_instance_ctrl_t g_dac8_ctrl;
dac_cfg_t g_dac8_cfg =
{
    .channel            = 0U,
    .ad_da_synchronized = false,
    .p_extend          = &g_dac8_cfg_extend
};
void basic_example (void)
{
```

```
fsp_err_t err;
   uint16_t value;
/* Pin configuration example: Output enable DA8_0(RA2A1) as Analog. */
/* Initialize the DAC8 channel */
   err = R_DAC8_Open(&g_dac8_ctrl, &g_dac8_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
   value = (uint8_t) DAC8_EXAMPLE_VALUE_ABC;
/* Write value to DAC module */
   err = R_DAC8_Write(&g_dac8_ctrl, value);
   assert(FSP_SUCCESS == err);
/* Start DAC8 conversion */
   err = R_DAC8_Start(&g_dac8_ctrl);
   assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | dac8_instance_ctrl_t |
| struct | dac8_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | dac8_mode_t |

## Data Structure Documentation

### ◆ dac8_instance_ctrl_t

| struct dac8_instance_ctrl_t |
|---|
| DAC8 instance control block. DO NOT INITIALIZE. |

### ◆ dac8_extended_cfg_t

| struct dac8_extended_cfg_t | | |
|---|---|---|
| DAC8 extended configuration | | |
| Data Fields | | |
| bool | enable_charge_pump | Enable DAC charge pump. |
| dac8_mode_t | dac_mode | DAC mode. |

## Enumeration Type Documentation

### ◆ dac8_mode_t

| enum dac8_mode_t | |
|---|---|
| Enumerator | |
| DAC8_MODE_NORMAL | DAC Normal mode. |
| DAC8_MODE_REAL_TIME | DAC Real-time (event link) mode. |

## Function Documentation

### ◆ R_DAC8_Open()

fsp_err_t R_DAC8_Open ( dac_ctrl_t *const *p_ctrl*, dac_cfg_t const *const *p_cfg* )

Perform required initialization described in hardware manual.

Implements dac_api_t::open.

Configures a single DAC channel. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | One or both of the following parameters may be NULL: p_ctrl or p_cfg |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | An invalid channel was requested. |
| FSP_ERR_NOT_ENABLED | Setting DACADSCR is not enabled when ADCSR.ADST = 0. |

***Note***

*This function is reentrant for different channels. It is not reentrant for the same channel.*

### ◆ R_DAC8_Write()

fsp_err_t R_DAC8_Write ( dac_ctrl_t *const *p_ctrl*, uint16_t *value* )

Write data to the D/A converter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |
| FSP_ERR_OVERFLOW | Data overflow when data value exceeds 8-bit limit. |

### ◆ R_DAC8_Start()

fsp_err_t R_DAC8_Start ( dac_ctrl_t *const *p_ctrl*)

Start the D/A conversion output.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is started successfully. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |
| FSP_ERR_IN_USE | Attempt to re-start a channel. |

### ◆ R_DAC8_Stop()

fsp_err_t R_DAC8_Stop ( dac_ctrl_t *const *p_ctrl*)

Stop the D/A conversion and disable the output signal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The control is successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_instance_ctrl has not been opened. |

### ◆ R_DAC8_Close()

| fsp_err_t R_DAC8_Close ( dac_ctrl_t *const *p_ctrl*) |
|---|
| Stop the D/A conversion, stop output, and close the DAC channel.<br><br>**Return values**<br>FSP_SUCCESS — The channel is successfully closed.<br>FSP_ERR_ASSERTION — p_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — Channel associated with p_instance_ctrl has not been opened. |

#### 5.2.1.9 DAC_B (r_dac_b)

Modules » Analog

**Functions**

| | |
|---|---|
| fsp_err_t | R_DAC_B_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg) |
| fsp_err_t | R_DAC_B_Write (dac_ctrl_t *p_api_ctrl, uint16_t value) |
| fsp_err_t | R_DAC_B_Start (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_B_Stop (dac_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_DAC_B_Close (dac_ctrl_t *p_api_ctrl) |

**Detailed Description**

Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface.

# Overview

### Features

The DAC_B module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers

### Build Time Configurations for r_dac_b

The following build time configurations are defined in fsp_cfg/r_dac_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Analog > DAC (r_dac_b)**

This module can be added to the Stacks tab via New Stack > Analog > DAC (r_dac_b). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_dac_b0 | Module name. |
| Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the hardware channel. |
| DAC Operating Voltage Mode | • Normal voltage mode (VREFH >= 2.7v)<br>• Low voltage mode (VREFH < 2.7v) | Normal voltage mode (VREFH >= 2.7v) | Select VREFH voltage for optimization of DAC output. |
| Data Format | • Right Justified<br>• Left Justified | Right Justified | Specify the DAC data format. |
| Internal Output | MCU Specific Options | | Enable DAC output to internal modules. |
| ELC Trigger Source | MCU Specific Options | | ELC event source that will trigger the DAC to start a conversion. |

**Clock Configuration**

The DAC peripheral module uses PCLKB as its clock source.

**Pin Configuration**

The DAn pins are used as analog outputs. Each DAC channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply pins for the DAC and ADC.

The VREFH and VREFL pins are top and ground voltage reference pins for the DAC and ADC.

# Usage Notes

**Output to Internal Modules**

The DAC output can be used as an analog input to other peripherals on the MCU (eg. ACMPHS, ADC) without outputting the voltage to an external pin.

This functionality must be enabled during configuration using dac_b_extended_cfg_t::internal_output_enabled. When internal output is enabled, the DAC output will be routed to internal modules. If the internal output is disabled, the output will be routed to the DAC output pin (DAn).

# Examples

### Basic Example

This is a basic example of minimal use of the R_DAC_B in an application. This example shows how this driver can be used for basic Digital to Analog Conversion operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA0 as Analog. */
 /* Initialize the DAC channel */
    err = R_DAC_B_Open(&g_dac_b_ctrl, &g_dac_b_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    value = (uint16_t) DAC_EXAMPLE_VALUE;
    err = R_DAC_B_Write(&g_dac_b_ctrl, value);
    assert(FSP_SUCCESS == err);
    err = R_DAC_B_Start(&g_dac_b_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | dac_b_instance_ctrl_t |
| struct | dac_b_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | dac_b_vrefh_t |

## Data Structure Documentation

### ◆ dac_b_instance_ctrl_t

| struct dac_b_instance_ctrl_t |
|---|
| DAC instance control block. |

### ◆ dac_b_extended_cfg_t

| struct dac_b_extended_cfg_t | | |
|---|---|---|
| DAC extended configuration | | |
| Data Fields | | |
| bool | internal_output_enabled | Internal output enable available on selected MCUs. |
| dac_data_format_t | data_format | Data format. |
| dac_b_vrefh_t | vrefh | High voltage reference range. |

## Enumeration Type Documentation

### ◆ dac_b_vrefh_t

| enum dac_b_vrefh_t | |
|---|---|
| DAC Vref Type | |
| Enumerator | |
| DAC_B_VREFH_NORMAL | VREFH >= 2.7v. |
| DAC_B_VREFH_LOW | VREFH < 2.7v. |

## Function Documentation

### ◆ R_DAC_B_Open()

fsp_err_t R_DAC_B_Open ( dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg* )

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. One or both of the following parameters may be NULL: p_api_ctrl or p_cfg<br>2. data_format value in p_cfg is out of range.<br>3. Extended configuration structure is set to NULL |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel ID requested in p_cfg may not available on the devices. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |

### ◆ R_DAC_B_Write()

fsp_err_t R_DAC_B_Write ( dac_ctrl_t * *p_api_ctrl*, uint16_t *value* )

Write data to the D/A converter and enable the output if it has not been enabled.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_B_Start()

fsp_err_t R_DAC_B_Start ( dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output if it has not been started.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is started successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_IN_USE | Attempt to re-start a channel. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_B_Stop()

fsp_err_t R_DAC_B_Stop ( dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The control is successfully stopped. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

### ◆ R_DAC_B_Close()

fsp_err_t R_DAC_B_Close ( dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Channel associated with p_ctrl has not been opened. |

## 5.2.1.10 Delta-Sigma Modulator Interface (r_dsmif)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_DSMIF_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_DSMIF_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend) |
| fsp_err_t | R_DSMIF_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_DSMIF_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask) |
| fsp_err_t | R_DSMIF_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_DSMIF_ScanStatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_DSMIF_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_DSMIF_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_DSMIF_Calibrate (adc_ctrl_t *p_ctrl, void const *p_extend) |
| fsp_err_t | R_DSMIF_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| fsp_err_t | R_DSMIF_CallbackSet (adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_DSMIF_Close (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_DSMIF_InfoGet (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info) |
| fsp_err_t | R_DSMIF_ErrorDetectionEnable (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_DSMIF_ErrorStatusGet (adc_ctrl_t *p_ctrl, dsmif_error_status_t *p_error_status) |

### Detailed Description

Driver for the DSMIF peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

The Delta-Sigma Modulator interface (DSMIF) functions similarly to a normal ADC, however, instead

of sampling an analog signal directly, DSMIF samples a PDM signal that is generated by an externally connected 1-bit Delta-Sigma Modulator.

dsmif-overview-light.drawio.svg
Figure 168: DSMIF Overview

Every instance of the DSMIF peripheral has 3 separate channels that can sample independent PDM signals. Each channel has a pair of sinc filters. One sinc filter is used for normal measurements, and the other sinc filter is used for detecting error conditions. The sinc filter configuration will determine the sample resolution, and the final sample rate for each channel. There are three primary configuration settings for the sinc filters:

- Sample Rate - The frequency that the PDM signal is sampled.
- Decimation Count - Number of samples that are removed between the integration stage and the differentiation stage of the sinc filter.
- Filter order - The filter order of the sinc filter: [1st, 2nd, 3rd] order.

**Features**

The DSMIF module supports the following features:

- Two independent units (3 channels per unit for a total of 6 channels)
- Configurable sinc filter ($1^{st}$, $2^{nd}$, $3^{rd}$ order)
- Configurable decimation count in the range (4,256)
- Support for master and slave modes
- Channel synchronization
- Trigger data capture via ELC events
- Trigger reset of decimation counter via ELC events
- Configurable error detection
  - Overcurrent error detection (Can trigger common error interrupt)
  - Overcurrent sum error detection (Can trigger common error interrupt)
  - Short-circuit detection (Can trigger common error interrupt)
  - Window notification error detection (Can trigger Port Output Enable for GPT (r_poeg))

# Configuration

**Build Time Configurations for r_dsmif**

The following build time configurations are defined in fsp_cfg/r_dsmif_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Error Detection Support | • Enabled<br>• Disabled | Enabled | If selected code for parameter checking is included in the build. |

### Configurations for Analog > DSMIF (r_dsmif)

This module can be added to the Stacks tab via New Stack > Analog > DSMIF (r_dsmif). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_dsmif0 | Module name |
| Clock Selection | • PCLKA<br>• DSMIFCLK<br>• GPTCLK | PCLKA | Select the DSMIF Core Clock. |
| Unit | • Unit 0<br>• Unit 1 | Unit 0 | DSMIF unit number. |
| Alignment | • Left Justified<br>• Right Justified | Right Justified | Select if the conversion result is aligned to the left or right of the channel data registers. |
| Callback | Name must be a valid C symbol | NULL | Callback for handling DSMIF interrupts. |
| Overcurrent Sum Error | | | |
| Overcurrent Sum Error > Upper Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent sum error upper limit detection. |
| Threshold | Must be an integer in the range [0,0x3FFFF]. | 0 | Overcurrent sum error detection upper limit. |
| Overcurrent Sum Error > Lower Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent sum error lower limit detection. |
| Threshold | Must be an integer in the range [0,0x3FFFF]. | 0 | Overcurrent sum error detection lower limit. |
| Channel Selection | • Channels 0, 1 and 2<br>• Channels 0 and 1<br>• Channel 0<br>• Channel 1<br>• Channel 2 | Channels 0, 1 and 2 | Specify the channel(s) used for detecting overcurrent sum errors. |
| Interrupts | | | |
| Channel Synchronization | • Channels are not | Channels are not synchronized | Select the channel synchronization mode. |

| | | | |
|---|---|---|---|
| | synchronized<br>• Channels 0 and 1 are synchronized<br>• Channels 0, 1 and 2 are synchronized | | Two or three channels that share the same configuration may optionaly configure a common interrupts instead of channel specific interrupts. |
| Common Data Update Interrupt Priority (CDUPD_COM) | MCU Specific Options | | The Common Data Update Interrupt occurs when a conversion is completed on all synchronized channels. |
| Common Capture A Data Update Interrupt Priority (CDACUPD_COM) | MCU Specific Options | | The Common Capture A Interrupt occurs when a Capture A conversion is completed on all synchronized channels. |
| Common Capture B Data Update Interrupt Priority (CDBCUPD_COM) | MCU Specific Options | | The Common Capture B Interrupt occurs when a Capture B conversion is completed on all synchronized channels. |
| Error Interrupt Priority | MCU Specific Options | | The Error Interrupt occurs when a short-cicuit, overcurrent, overcurrent sum, or window notification error occurs on any channel. |

**Configurations for Analog > DSMIF Channel Configuration (r_dsmif)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_dsmif_channel0 | Module name |
| Clock | | | |
| Clock Direction | • Slave Mode<br>• Master Mode | Slave Mode | Select the conversion clock mode for the DSMIF channel. In master mode the clock is output from the DSMIF channel. In slave mode the clock is input to the DSMIF channel. |
| Clock Edge | • Negative Edge<br>• Positive Edge | Negative Edge | Select the sampling edge of DSMCLKn for |

| | | | |
|---|---|---|---|
| | | | this channel. |
| CKDIV Setting (Master Mode Only) | Must be an integer in the range [2,63]. | 2 | Set the A/D conversion clock division ratio. The clock division ratio is equal to 2 x (CKDIV + 1). This setting is only used in master mode. |
| DSMCLK Frequency (Slave Mode Only) | Must be an integer in the range [0,25000000]. | 25000000 | If slave mode is selected, then manually enter the DSMCLK frequency. This setting is used to calculate the settling time of the DSMIF sinc filters. |
| Filter | | | |
| Filter Order | • First Order<br>• Second Order<br>• Third Order | Third Order | Select 1st, 2nd, or 3rd order sinc filter for current data. |
| Decimation Count | Must be an integer in the range [4,256]. | 256 | Decimation ratio of the current data sinc filter. |
| Trigger | | | |
| Trigger > Filter Initialization | | | |
| Trigger | • No Trigger<br>• Trigger 0 (ELC_DSMIFm_CNT0)<br>• Trigger 1 (ELC_DSMIFm_CNT1)<br>• Trigger 2 (ELC_DSMIFm_CNT2) | No Trigger | Select filter initialization trigger. The filter initialization trigger resets the sinc filter's decimation clock via ELC event. |
| Edge | • Negative<br>• Positive | Positive | Select either a positive or negative edge for the filter initialization trigger. |
| Capture Trigger A | • No Trigger<br>• Trigger 0 (ELC_DSMIFm_CAP0)<br>• Trigger 1 (ELC_DSMIFm_CAP1)<br>• Trigger 2 (ELC_DSMIFm_CAP2)<br>• Trigger 3 (ELC_DSMIFm_CAP3)<br>• Trigger 4 (ELC_DSMIFm_CAP4)<br>• Trigger 5 (ELC_DSMIFm_CAP5) | No Trigger | Current capture trigger A selection. |
| Capture Trigger B | • No Trigger | No Trigger | Current capture trigger |

| | | | |
|---|---|---|---|
| | • Trigger 0 (ELC_DSMIFm_CAP0)<br>• Trigger 1 (ELC_DSMIFm_CAP1)<br>• Trigger 2 (ELC_DSMIFm_CAP2)<br>• Trigger 3 (ELC_DSMIFm_CAP3)<br>• Trigger 4 (ELC_DSMIFm_CAP4)<br>• Trigger 5 (ELC_DSMIFm_CAP5) | | B selection. |
| Error Detection | | | |
| Error Detection > Filter | | | |
| Filter Order | • First Order<br>• Second Order<br>• Third Order | Third Order | Select 1st, 2nd, or 3rd order sinc filter for overcurrent detection. |
| Decimation Count | Must be an integer in the range [4,256]. | 256 | Decimation ratio of the overcurrent detection filter. |
| Error Detection > Overcurrent Detection 0 | | | |
| Error Detection > Overcurrent Detection 0 > Upper Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 0 upper limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0xFFFF | Configure the overcurrent 0 upper limit. Valid settings are in the range [0,65535]. |
| Error Detection > Overcurrent Detection 0 > Lower Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 0 lower limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0 | Configure the overcurrent 0 lower limit. Valid settings are in the range [0,65535]. |
| Error Detection > Overcurrent Detection 1 | | | |
| Error Detection > Overcurrent Detection 1 > Upper Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 1 upper limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0xFFFF | Configure the overcurrent 1 upper limit. Valid settings are |

| | | | |
|---|---|---|---|
| | | | in the range [0,65535]. |
| Error Detection > Overcurrent Detection 1 > Lower Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 1 lower limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0 | Configure the overcurrent 1 lower limit. Valid settings are in the range [0,65535]. |
| Error Detection > Overcurrent Detection 2 | | | |
| Error Detection > Overcurrent Detection 2 > Upper Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 2 upper limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0xFFFF | Configure the overcurrent 2 upper limit. Valid settings are in the range [0,65535]. |
| Error Detection > Overcurrent Detection 2 > Lower Limit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent 2 lower limit. |
| Threshold | Must be an integer in the range [0,65535]. | 0 | Configure the overcurrent 2 lower limit. Valid settings are in the range [0,65535]. |
| Error Detection > Overcurrent Window Notifcation 0 | | | |
| Notification | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent window notification 0. |
| Mode | • Overcurrent Data < Lower Limit 0 or Upper Limit 0 < Overcurrent Data<br>• Lower Limit 0 <= Overcurrent Data <= Upper Limit 0 | Overcurrent Data < Lower Limit 0 or Upper Limit 0 < Overcurrent Data | Select overcurrent window notification 0 mode. |
| Error Detection > Overcurrent Window Notifcation 1 | | | |
| Notification | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent window notification 1. |
| Mode | • Overcurrent | Overcurrent Data < | Select overcurrent |

| | | | |
|---|---|---|---|
| | Data < Lower Limit 1 or Upper Limit 1 < Overcurrent Data<br>• Lower Limit 1 <= Overcurrent Data <= Upper Limit 1 | Lower Limit 1 or Upper Limit 1 < Overcurrent Data | window notification 1 mode. |
| Error Detection > Overcurrent Window Notifcation 2 | | | |
| Notification | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent window notification 2. |
| Mode | • Overcurrent Data < Lower Limit 2 or Upper Limit 2 < Overcurrent Data<br>• Lower Limit 2 <= Overcurrent Data <= Upper Limit 2 | Overcurrent Data < Lower Limit 2 or Upper Limit 2 < Overcurrent Data | Select overcurrent window notification 2 mode. |
| Error Detection > Overcurrent Window Notifcation 3 | | | |
| Notification | • Disabled<br>• Enabled | Disabled | Enable or disable overcurrent window notification 3. |
| Mode | Refer to the RA Configuration tool for available options. | Window notification 0 | Select overcurrent window notification 3 mode. |
| Error Detection > Short Circuit | | | |
| Detection | • Disabled<br>• Enabled | Disabled | Enable or disable short circuit detection. |
| Low Count | Must be an integer in the range [0,8191]. | 0 | Short circuit detection low count. |
| High Count | Must be an integer in the range [0,8191]. | 0 | Short circuit detection high count. |
| Interrupts | | | |
| Current Data Update Interrupt Priority (CDUPD) | MCU Specific Options | | The Current Data Update Interrupt occurs when a conversion is completed. |
| Capture A Data Update Interrupt Priority (CDAUPD) | MCU Specific Options | | The Capture A Data Update Interrupt occurs when a capture A conversion is completed. |

| | | |
|---|---|---|
| Capture B Data Update Interrupt Priority (CDBUPD) | MCU Specific Options | The Capture B Data Update Interrupt occurs when a capture B conversion is completed. |

### Clock Configuration

The DSMIF conversion rate is determined by the **DSMIF Core Clock**, **A/D Conversion Division Ratio (CKDIV)**, and the configured sinc filter's **decimation count**.

The **DSMIF Core Clock** is common to all channels and may be configured to the following settings:

- PCLKA - Peripheral Clock A
- DSMIFCLK - Dedicated DSMIF Clock
- GPTCLK - Dedicated GPT Clock

The **A/D Conversion Division Ratio (CKDIV)** is a channel specific setting that further divides the core clock by the following formula:

```
A/D Conversion Division Ratio = (2 x (CKDIV + 1))
```

*Note*

1. *CKDIV is set in the range [2,63].*
2. *CKDIV is only valid in master mode.*

The **decimation count** is a channel specific sinc filter setting that will determine the final conversion rate of a DSMIF channel.

*Note*

1. *The decimation count is set in the range [4,256].*

### DSMIF Recommended Clock Configuration:

| DSMIF Core Clock (Mhz) | CKDIV | DSMCLKn (Mhz) |
|---|---|---|
| 300 | 0x05 | 25 |
| | 0x08 | 16.67 |
| | 0x09 | 15 |
| | 0x0E | 10 |
| | 0x1D | 5 |
| 200 | 0x03 | 25 |
| | 0x04 | 20 |
| | 0x07 | 12.5 |
| | 0x09 | 10 |
| | 0x0F | 6.25 |

| | |
|---|---|
| 0x13 | 5 |

*Note*

*1. If settings other than the recommended are used, there is an additional clock constraint: DSMIF Core Clock/5 ? PCLKA*

### Pin Configuration

Each DSMIF channel has two pins:

- DSMmCLKn - DSMIF Clock Input/Output. In Master mode the clock pin is an output, in slave mode, the clock is an input.
- DSMmDATn - DSMIF Data Input.

## Usage Notes

### Channel Synchronization

Multiple channels may be optionally synchronized in order to generate common interrupts in response to the following events:

- ADC_EVENT_CONVERSION_COMPLETE
- ADC_EVENT_CAPTURE_A
- ADC_EVENT_CAPTURE_B

The following sets of channels may be synchronized:

- Channel 0 and Channel 1
- Channel 0, Channel 1, and Channel 2

When synchronizing channels, each channel must use the same 'Clock', 'Filter', and 'Trigger' configuration settings. For convenience, channel configurations can be reused between channels of a DSMIF instance instead of creating new channel configurations for each channel.

*Note*

*1. Channel configurations cannot be reused between different DSMIF instances.*
*2. Channel configurations can only be reused between channels that are synchronized.*

### Filter Settling Time

The filter settling time is the time required for the filter results to be valid after the DSMIF channels have been started. The settling time is determined by the DSMCLK period, Decimation Ratio, Filter Order, DSMIF Core Clock period, and Peripheral Clock period.

The following macros are generated by the tooling in order to assist with determining the required settling time:

```
/* Calculated settling time required for DSMIF filters (See "Settling Time" section
in the MCU user manual). */
#define DSMIF0_FILTER_SETTLING_TIME_SYNCHRONOUS_US (...)
#define DSMIF0_FILTER_SETTLING_TIME_ASYNCHRONOUS_US (...)
```

```
#define DSMIF0_FILTER_SETTLING_TIME_GPT_SYNCHRONOUS_US (...)
#define DSMIF0_FILTER_SETTLING_TIME_GPT_ASYNCHRONOUS_US (...)
```

*Note*

*1. For more information on how to calculate the settling time refer to the MCU user manual.*

### Error Detection

There are four different types of error interrupts generated by the DSMIF peripheral:

- **Overcurrent error detection** - Triggered when a conversion result exceeds a configured upper/lower limit on a particular channel.
- **Overcurrent sum error detection** - Triggered when the sum of conversion results from a set of channels exceeds a configured upper/lower limit.
- **Window Notification error detection** - Triggered when a conversion result falls within a set of configurable ranges.
- **Short-circuit detection** - Triggered when a configured number of consecutive '1's or '0's are input on a DSMmDATn pin.

In order to enable error detection, the application must call R_DSMIF_ErrorDetectionEnable after waiting for all DSMIF channel filters to stabilize. The amount of time required for a filter to stabilize is dependent on the filter parameters, and clock settings. Refer to the MCU user manual for determining the filter stabilization time.

Once error detection is enabled, R_DSMIF_ErrorStatusGet can be called to read the error status.

*Note*

*1. Error detection is disabled when channel operation is stopped via R_DSMIF_ScanStop. When starting channel operation again, the same settling time is required before reenabling error detection.*

### Error Interrupt

If the error interrupt is enabled, the following configured error flags will generate an interrupt request:

- Overcurrent error detection
- Overcurrent sum error detection
- Short-circuit detection

*Note*

*1. Since the error interrupt is a level interrupt, R_DSMIF_ErrorStatusGet must be called from the user callback in order to clear the associated error status flags.*
*2. Window Notification error detection does not generate an interrupt request.*

### Limitations

Developers should be aware of the following limitations when using DSMIF:

- The DSMIF peripheral must be stopped prior to entering software standby mode.

# Examples

### Basic Example

This is a basic example of minimal use of the ADC in an application.

```
void dsmif_basic_example (void)
{
 fsp_err_t err = R_DSMIF_Open(&g_dsmif_ctrl, &g_dsmif_cfg);
    assert(FSP_SUCCESS == err);
    err = R_DSMIF_ScanStart(&g_dsmif_ctrl);
    assert(FSP_SUCCESS == err);
 /* The application must wait for the configured filters to stabilize. */
    app_wait_for_filter_stabilization(DSMIF0_FILTER_SETTLING_TIME_SYNCHRONOUS_US);
    err = R_DSMIF_ErrorDetectionEnable(&g_dsmif_ctrl);
    assert(FSP_SUCCESS == err);
}
void dsmif_callback (adc_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case ADC_EVENT_CONVERSION_COMPLETE:
     {
           uint32_t conversion;
 fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
 if (FSP_SUCCEESS == err)
     {
 /* Application process the converted data. */
               app_dsmif_data_process(p_args->channel, conversion);
           }
 break;
      }
 case ADC_EVENT_CAPTURE_A:
     {
           uint32_t conversion;
 fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
```

```
if (FSP_SUCCEESS == err)
     {
/* Application process the converted capture a data. */
               app_dsmif_capture_a_data_process(p_args->channel, conversion);
           }
break;
      }
case ADC_EVENT_CAPTURE_B:
     {
           uint32_t conversion;
fsp_err_t err = R_DSMIF_Read32(&g_dsmif_ctrl, p_args->channel, &conversion);
if (FSP_SUCCEESS == err)
     {
/* Application process the converted capture b data. */
               app_dsmif_capture_b_data_process(p_args->channel, conversion);
           }
break;
      }
case ADC_EVENT_CONVERSION_ERROR:
     {
dsmif_error_status_t error_status;
fsp_err_t            err = R_DSMIF_ErrorStatusGet(&g_dsmif_ctrl, &error_status);
if (FSP_SUCCESS == err)
     {
/* Application process the error status. */
               app_dsmif_error_process(&error_status);
           }
break;
      }
default:
     {
/* Invalid event. */
break;
      }
```

```
    }
}
```

### Reading Current/Overcurrent/Capture data

This is a basic example of how to read the following types of data by OR'ing the appropriate channel mask:

- Current Data
- Capture A/B Data
- Overcurrent Data
- Overcurrent Capture Data
- Overcurrent Sum Capture Data

```
void dsmif_channel_data_masks_example (void)
{
 fsp_err_t err;
    uint16_t conversion;
 /* Read normal current data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0, &conversion);
    assert(FSP_SUCCESS == err);
 /* Read capture current data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_CAPTURE_A, &conversion);
    assert(FSP_SUCCESS == err);
 /* Read overcurrent data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT, &conversion);
    assert(FSP_SUCCESS == err);
 /* Read overcurrent capture data 0 from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT0_CAPTURE,
&conversion);
    assert(FSP_SUCCESS == err);
 /* Read overcurrent sum capture data from channel 0. */
    err = R_DSMIF_Read32(&g_dsmif_ctrl, ADC_CHANNEL_0_OVERCURRENT_SUM_CAPTURE,
&conversion);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | dsmif_clock_cfg_t |
| struct | dsmif_filter_cfg_t |
| struct | dsmif_trigger_cfg_t |
| struct | dsmif_interrupt_cfg_t |
| struct | dsmif_error_detection_cfg_t |
| struct | dsmif_error_status_t |
| struct | dsmif_channel_cfg_t |
| struct | dsmif_extended_cfg_t |
| struct | dsmif_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | dsmif_clock_selection_t |
| enum | dsmif_overcurrent_sum_error_channel_t |
| enum | dsmif_channel_synchronization_mode_t |
| enum | dsmif_clock_direction_t |
| enum | dsmif_clock_edge_t |
| enum | dsmif_filter_order_t |
| enum | dsmif_data_shift_t |
| enum | dsmif_capture_trigger_t |
| enum | dsmif_filter_initialization_trigger_t |
| enum | dsmif_overcurrent_window_notification_mode_t |
| enum | dsmif_overcurrent_window_notification_mode3_t |
| enum | dsmif_overcurrent_status_t |
| enum | dsmif_short_circuit_status_t |
| enum | dsmif_overcurrent_sum_status_t |
| enum | dsmif_overcurrent_window_notification_status_t |

## Data Structure Documentation

### ◆ dsmif_clock_cfg_t

| struct dsmif_clock_cfg_t | | |
|---|---|---|
| Channel clock configuration | | |
| Data Fields | | |
| dsmif_clock_direction_t | direction | A/D conversion clock master/slave switching. |
| dsmif_clock_edge_t | edge | Sampling edge selection. |
| uint8_t | divider | A/D conversion clock division ratio. |

### ◆ dsmif_filter_cfg_t

| struct dsmif_filter_cfg_t | | |
|---|---|---|
| Filter configuration | | |
| Data Fields | | |
| dsmif_filter_order_t | filter_order | Sinc filter order (1st, 2nd, or 3rd) |
| dsmif_data_shift_t | data_shift | Data shift setting. |
| uint8_t | decimation_count | Decimation count [0,255]. |

### ◆ dsmif_trigger_cfg_t

| struct dsmif_trigger_cfg_t | | |
|---|---|---|
| Event trigger configuration. | | |
| Data Fields | | |
| dsmif_capture_trigger_t | capture_trigger_a | Current capture trigger A selection. |
| dsmif_capture_trigger_t | capture_trigger_b | Current capture trigger B selection. |
| dsmif_filter_initialization_trigger_t | filter_initialization_trigger | Current measurement filter initialization trigger selection. |
| dsmif_clock_edge_t | filter_initialization_edge | Current measurement filter initialization trigger edge selection. |

### ◆ dsmif_interrupt_cfg_t

| struct dsmif_interrupt_cfg_t |
|---|
| Channel interrupt configuration. |
| Data Fields |

| | | |
|---|---|---|
| IRQn_Type | current_data_update_irq | Channel current data update interrupt request. |
| uint8_t | current_data_update_ipl | Channel current data update interrupt priority. |
| IRQn_Type | capture_data_a_irq | Channel capture data a interrupt request. |
| uint8_t | capture_data_a_ipl | Channel capture data a interrupt priority. |
| IRQn_Type | capture_data_b_irq | Channel capture data b interrupt request. |
| uint8_t | capture_data_b_ipl | Channel capture data b interrupt priority. |

### ◆ dsmif_error_detection_cfg_t

struct dsmif_error_detection_cfg_t

Channel error configuration:

- Overcurrent upper/lower limit error detection 0,1,2
- Overcurrent window notification 0,1,2,3
- Short circuit low/high detection

Data Fields

| | | |
|---|---|---|
| union dsmif_error_detection_cfg_t | __unnamed__ | |
| union dsmif_error_detection_cfg_t | __unnamed__ | |
| union dsmif_error_detection_cfg_t | __unnamed__ | |
| bool | short_circuit_detection_enable | Enable short circuit detection. |
| uint32_t | short_circuit_detection_low_count | Short circuit low count. |
| uint32_t | short_circuit_detection_high_count | Short circuit high count. |
| uint32_t | overcurrent_lower_limit[3] | Overcurrent lower limit settings. |
| uint32_t | overcurrent_upper_limit[3] | Overcurrent upper limit settings. |
| dsmif_filter_cfg_t | overcurrent_filter_cfg | Sinc filter configuration for overcurrent detection. |

### ◆ dsmif_error_status_t

struct dsmif_error_status_t

Error status.

| Data Fields | | |
|---|---|---|
| dsmif_overcurrent_status_t | channel_overcurrent_status | Channel Overcurrent state. |
| dsmif_short_circuit_status_t | channel_short_circuit_status | Channel Short circuit state. |
| dsmif_overcurrent_sum_status_t | overcurrent_sum_status | Overcurrent Sum state. |
| dsmif_overcurrent_window_notification_status_t | channel_overcurrent_window_status | Channel Overcurrent Window State. |

### ◆ dsmif_channel_cfg_t

| struct dsmif_channel_cfg_t | | |
|---|---|---|
| Channel configuration. | | |
| Data Fields | | |
| dsmif_clock_cfg_t | clock_cfg | Channel clock configuration. |
| dsmif_filter_cfg_t | filter_cfg | Channel sinc filter configuration. |
| dsmif_trigger_cfg_t | trigger_cfg | Channel event trigger configuration. |
| dsmif_interrupt_cfg_t | irq_cfg | Channel interrupt configuration. |

### ◆ dsmif_extended_cfg_t

| struct dsmif_extended_cfg_t | | |
|---|---|---|
| DSMIF configuration extension. This extension is required and must be provided in dsmif_cfg_t::p_extend. | | |
| Data Fields | | |
| dsmif_clock_selection_t | clock_selection | DSMIF core clock selection. |
| uint32_t | channel_mask | Channel bitmask. |
| dsmif_channel_cfg_t const * | p_channel_cfgs[DSMIF_MAX_NUM_CHANNELS] | Configuration for each channel, set to NULL if unused. |
| dsmif_channel_synchronization_mode_t | synchronization_mode | Channel synchronization mode. |
| IRQn_Type | common_current_data_irq | Common current data interrupt request. |
| uint8_t | common_current_data_ipl | Common current data interrupt priority. |
| IRQn_Type | common_capture_data_a_irq | Channel capture data a interrupt request. |
| uint8_t | common_capture_data_a_ipl | Channel capture data a interrupt priority. |
| IRQn_Type | common_capture_data_b_irq | Channel capture data b interrupt request. |

| | | |
|---|---|---|
| uint8_t | common_capture_data_b_ipl | Channel capture data b interrupt priority. |

### ◆ dsmif_instance_ctrl_t

| struct dsmif_instance_ctrl_t |
|---|
| DSMIF instance control block. DO NOT INITIALIZE. |

## Enumeration Type Documentation

### ◆ dsmif_clock_selection_t

| enum dsmif_clock_selection_t | |
|---|---|
| DSMIF clock selection. | |
| Enumerator | |
| DSMIF_CLOCK_SELECTION_PCLKA | Select PCLKA as the DSMIF clock. |
| DSMIF_CLOCK_SELECTION_DSMIFCLK | Select DSMIFCLK as the DSMIF clock. |
| DSMIF_CLOCK_SELECTION_GPTCLK | Select GPTCLK as the DSMIF clock. |

### ◆ dsmif_overcurrent_sum_error_channel_t

| enum dsmif_overcurrent_sum_error_channel_t | |
|---|---|
| Select the channel(s) for overcurrent sum error detection. | |
| Enumerator | |
| DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0_2 | Detects error of sum value of overcurrent data of CH0, CH1, CH2. |
| DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0_1 | Detects error of sum value of overcurrent data of CH0, CH1. |
| DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_0 | Detects error of overcurrent data of CH0. |
| DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_1 | Detects error of overcurrent data of CH1. |
| DSMIF_OVERCURRENT_SUM_ERROR_CHANNEL_2 | Detects error of overcurrent data of CH2. |

### ◆ dsmif_channel_synchronization_mode_t

| enum dsmif_channel_synchronization_mode_t | |
|---|---|
| Select the channel(s) that are synchronized. | |
| Enumerator | |
| DSMIF_CHANNEL_SYNCHRONIZATION_MODE_DISABLED | Channels are not synchronized. |
| DSMIF_CHANNEL_SYNCHRONIZATION_MODE_CHANNEL_0_2 | Channel 0 and Channel 1 and Channel 2 are synchronized. |
| DSMIF_CHANNEL_SYNCHRONIZATION_MODE_CHANNEL_0_1 | Channel 0 and Channel 1 are synchronized. |

### ◆ dsmif_clock_direction_t

| enum dsmif_clock_direction_t | |
|---|---|
| Enumerator | |
| DSMIF_CLOCK_DIRECTION_SLAVE | Clock signal is input (slave operation) |
| DSMIF_CLOCK_DIRECTION_MASTER | Clock signal is output (master operation) |

### ◆ dsmif_clock_edge_t

| enum dsmif_clock_edge_t | |
|---|---|
| Select the clock edge for sampling DSMIF data. | |
| Enumerator | |
| DSMIF_CLOCK_EDGE_NEGATIVE | Data is sampled on the negative edge of clock signal. |
| DSMIF_CLOCK_EDGE_POSITIVE | Data is sampled on the positive edge of clock signal. |

### ◆ dsmif_filter_order_t

| enum dsmif_filter_order_t | |
|---|---|
| Select the DSMIF filter order. | |
| Enumerator | |
| DSMIF_FILTER_ORDER_3RD | Filter order setting 3rd order. |
| DSMIF_FILTER_ORDER_1ST | Filter order setting 1st order. |
| DSMIF_FILTER_ORDER_2ND | Filter order setting 2nd order. |

### ◆ dsmif_data_shift_t

| enum dsmif_data_shift_t | |
|---|---|
| Select the [MSB,LSB] that are copied from the filter output into the data register. | |
| Enumerator | |
| DSMIF_DATA_SHIFT_23_8 | Data shift setting [23:8]. |
| DSMIF_DATA_SHIFT_22_7 | Data shift setting [22:7]. |
| DSMIF_DATA_SHIFT_21_6 | Data shift setting [21:6]. |
| DSMIF_DATA_SHIFT_20_5 | Data shift setting [20:5]. |
| DSMIF_DATA_SHIFT_19_4 | Data shift setting [19:4]. |
| DSMIF_DATA_SHIFT_18_3 | Data shift setting [18:3]. |
| DSMIF_DATA_SHIFT_17_2 | Data shift setting [17:2]. |
| DSMIF_DATA_SHIFT_16_1 | Data shift setting [16:1]. |
| DSMIF_DATA_SHIFT_15_0 | Data shift setting [15:0]. |
| DSMIF_DATA_SHIFT_14_0 | Data shift setting [14:0]. |
| DSMIF_DATA_SHIFT_13_0 | Data shift setting [13:0]. |
| DSMIF_DATA_SHIFT_12_0 | Data shift setting [12:0]. |
| DSMIF_DATA_SHIFT_11_0 | Data shift setting [11:0]. |
| DSMIF_DATA_SHIFT_10_0 | Data shift setting [10:0]. |
| DSMIF_DATA_SHIFT_9_0 | Data shift setting [9:0]. |
| DSMIF_DATA_SHIFT_8_0 | Data shift setting [8:0]. |
| DSMIF_DATA_SHIFT_7_0 | Data shift setting [7:0]. |
| DSMIF_DATA_SHIFT_6_0 | Data shift setting [6:0]. |
| DSMIF_DATA_SHIFT_5_0 | Data shift setting [5:0]. |
| DSMIF_DATA_SHIFT_4_0 | Data shift setting [4:0]. |

### ◆ dsmif_capture_trigger_t

| enum dsmif_capture_trigger_t | |
|---|---|
| Enumerator | |
| DSMIF_CAPTURE_TRIGGER_NONE | Do not capture. |
| DSMIF_CAPTURE_TRIGGER_0 | Select current data capture trigger 0. |
| DSMIF_CAPTURE_TRIGGER_1 | Select current data capture trigger 1. |
| DSMIF_CAPTURE_TRIGGER_2 | Select current data capture trigger 2. |
| DSMIF_CAPTURE_TRIGGER_3 | Select current data capture trigger 3. |
| DSMIF_CAPTURE_TRIGGER_4 | Select current data capture trigger 4. |
| DSMIF_CAPTURE_TRIGGER_5 | Select current data capture trigger 5. |

### ◆ dsmif_filter_initialization_trigger_t

| enum dsmif_filter_initialization_trigger_t | |
|---|---|
| Enumerator | |
| DSMIF_FILTER_INITIALIZATION_TRIGGER_NONE | Do not initialize. |
| DSMIF_FILTER_INITIALIZATION_TRIGGER_0 | Select decimation dividing counter initialization trigger 0. |
| DSMIF_FILTER_INITIALIZATION_TRIGGER_1 | Select decimation dividing counter initialization trigger 1. |
| DSMIF_FILTER_INITIALIZATION_TRIGGER_2 | Select decimation dividing counter initialization trigger 2. |

### ◆ dsmif_overcurrent_window_notification_mode_t

| enum dsmif_overcurrent_window_notification_mode_t | |
|---|---|
| Overcurrent window notification window mode setting [0,2]. | |
| Enumerator | |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE_OUTSIDE | Overcurrent data <= lower limit, upper limit <= overcurrent data. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE_INSIDE | lower limit < overcurrent data < upper limit. |

### ◆ dsmif_overcurrent_window_notification_mode3_t

| enum dsmif_overcurrent_window_notification_mode3_t | |
|---|---|
| Overcurrent window detection window mode setting for window notification 3. | |
| Enumerator | |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0 | Overcurrent detection window notification 0. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1 | Overcurrent detection window notification 1. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_2 | Overcurrent detection window notification 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_1 | Overcurrent detection window notification 0 OR 1. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_1 | Overcurrent detection window notification 0 AND 1. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_2 | Overcurrent detection window notification 0 OR 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_2 | Overcurrent detection window notification 0 AND 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1_OR_2 | Overcurrent detection window notification 1 OR 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_1_AND_2 | Overcurrent detection window notification 1 AND 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_OR_1_OR_2 | Overcurrent detection window notification 0 OR 1 OR 2. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_MODE3_0_AND_1_AND_2 | Overcurrent detection window notification 0 AND 1 AND 2. |

### ◆ dsmif_overcurrent_status_t

| enum dsmif_overcurrent_status_t | |
|---|---|
| DSMIF Channel Overcurrent detect status. | |
| Enumerator | |
| DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_0 | Channel 0 overcurrent lower limit detection 0 flag. |
| | |

| | |
|---|---|
| DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_0 | Channel 1 overcurrent lower limit detection 0 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_0 | Channel 2 overcurrent lower limit detection 0 flag. |
| DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_0 | Channel 0 overcurrent upper limit exceeded 0 flag. |
| DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_0 | Channel 1 overcurrent upper limit exceeded 0 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_0 | Channel 2 overcurrent upper limit exceeded 0 flag. |
| DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_1 | Channel 0 overcurrent lower limit detection 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_1 | Channel 1 overcurrent lower limit detection 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_1 | Channel 2 overcurrent lower limit detection 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_1 | Channel 0 overcurrent upper limit exceeded 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_1 | Channel 1 overcurrent upper limit exceeded 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_1 | Channel 2 overcurrent upper limit exceeded 1 flag. |
| DSMIF_OVERCURRENT_STATUS_CH0_LOWER_LIMIT_2 | Channel 0 overcurrent lower limit detection 2 flag. |
| DSMIF_OVERCURRENT_STATUS_CH1_LOWER_LIMIT_2 | Channel 1 overcurrent lower limit detection 2 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_LOWER_LIMIT_2 | Channel 2 overcurrent lower limit detection 2 flag. |
| DSMIF_OVERCURRENT_STATUS_CH0_UPPER_LIMIT_2 | Channel 0 overcurrent upper limit exceeded 2 flag. |
| DSMIF_OVERCURRENT_STATUS_CH1_UPPER_LIMIT_2 | Channel 1 overcurrent upper limit exceeded 2 flag. |
| DSMIF_OVERCURRENT_STATUS_CH2_UPPER_LIMIT_2 | Channel 2 overcurrent upper limit exceeded 2 flag. |

### ◆ dsmif_short_circuit_status_t

| enum dsmif_short_circuit_status_t | |
|---|---|
| DSMIF Channel Short circuit detect status. | |
| Enumerator | |
| DSMIF_SHORT_CIRCUIT_STATUS_CH0 | Channel 0 short circuit detection flag. |
| DSMIF_SHORT_CIRCUIT_STATUS_CH1 | Channel 1 short circuit detection flag. |
| DSMIF_SHORT_CIRCUIT_STATUS_CH2 | Channel 2 short circuit detection flag. |

### ◆ dsmif_overcurrent_sum_status_t

| enum dsmif_overcurrent_sum_status_t | |
|---|---|
| DSMIF Overcurrent Sum detect status. | |
| Enumerator | |
| DSMIF_OVERCURRENT_SUM_STATUS_LOWER_LIMIT | Overcurrent sum error lower limit detection flag. |
| DSMIF_OVERCURRENT_SUM_STATUS_UPPER_LIMIT | Overcurrent sum error upper limit detection flag. |

### ◆ dsmif_overcurrent_window_notification_status_t

| enum dsmif_overcurrent_window_notification_status_t | |
|---|---|
| DSMIF Overcurrent Window detect status. | |
| Enumerator | |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_0 | Channel 0 overcurrent detection window notification 0 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_0 | Channel 1 overcurrent detection window notification 0 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_0 | Channel 2 overcurrent detection window notification 0 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_1 | Channel 0 overcurrent detection window notification 1 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_1 | Channel 1 overcurrent detection window notification 1 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_1 | Channel 2 overcurrent detection window notification 1 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_2 | Channel 0 overcurrent detection window notification 2 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_2 | Channel 1 overcurrent detection window notification 2 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_2 | Channel 2 overcurrent detection window notification 2 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH0_3 | Channel 0 overcurrent detection window notification 3 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH1_3 | Channel 1 overcurrent detection window notification 3 flag. |
| DSMIF_OVERCURRENT_WINDOW_NOTIFICATION_STATUS_CH2_3 | Channel 2 overcurrent detection window notification 3 flag. |

## Function Documentation

### ◆ R_DSMIF_Open()

fsp_err_t R_DSMIF_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Applies power to the DSMIF and initializes the hardware based on the user configuration. As part of this initialization, set interrupts, set DSMIF error interrupt registers, etc.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Control block is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt is disabled |
| FSP_ERR_IP_UNIT_NOT_PRESENT | The Unit requested in the p_cfg parameter is not available on this device. |

### ◆ R_DSMIF_ScanCfg()

fsp_err_t R_DSMIF_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_extend* )

adc_api_t::scanCfg is not supported on DSMIF.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported by the DSMIF instance. |

### ◆ R_DSMIF_ScanStart()

fsp_err_t R_DSMIF_ScanStart ( adc_ctrl_t * *p_ctrl*)

Starts all configured DSMIF channels

***Note***

*After executing R_DSMIF_ScanStart, it is necessary to wait until the filter result is output stably.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started or hardware triggers enabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_ScanGroupStart()

fsp_err_t R_DSMIF_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_mask* )

adc_api_t::scanGroupStart is not supported on DSMIF.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported by the DSMIF instance. |

### ◆ R_DSMIF_ScanStop()

fsp_err_t R_DSMIF_ScanStop ( adc_ctrl_t * *p_ctrl*)

Stops any in-progress scan started by software.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped or hardware triggers disabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_TIMEOUT | Timeout error. |

### ◆ R_DSMIF_ScanStatusGet()

fsp_err_t R_DSMIF_ScanStatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Returns the status of a scan started by software.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | No software scan is in progress. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_Read()

fsp_err_t R_DSMIF_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads the most recent result from a channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_Read32()

fsp_err_t R_DSMIF_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads the most recent conversion result from a single channel into a 32-bit result.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_Calibrate()

fsp_err_t R_DSMIF_Calibrate ( adc_ctrl_t * *p_ctrl*, void const * *p_extend* )

adc_api_t::calibrate is not supported on DSMIF.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported by the DSMIF instance. |

### ◆ R_DSMIF_OffsetSet()

fsp_err_t R_DSMIF_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset* )

adc_api_t::offsetSet is not supported on DSMIF.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported by the DSMIF instance. |

### ◆ R_DSMIF_CallbackSet()

fsp_err_t R_DSMIF_CallbackSet ( adc_ctrl_t *const *p_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void *const *p_context*, adc_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_DSMIF_Close()

fsp_err_t R_DSMIF_Close ( adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the DSMIF peripheral.

***Note***

*This function is delayed at least until the outage is complete, as required by the DSMIF outage procedure.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_InfoGet()

fsp_err_t R_DSMIF_InfoGet ( adc_ctrl_t *const *p_ctrl*, adc_info_t *const *p_adc_info* )

adc_api_t::infoGet is not supported on DSMIF.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported by the DSMIF instance. |

### ◆ R_DSMIF_ErrorDetectionEnable()

fsp_err_t R_DSMIF_ErrorDetectionEnable ( adc_ctrl_t * *p_ctrl*)

Enable error detection.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration setting successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_DSMIF_ErrorStatusGet()

fsp_err_t R_DSMIF_ErrorStatusGet ( adc_ctrl_t * *p_ctrl*, dsmif_error_status_t * *p_error_status* )

Returns the error status of a scan started by software.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | No software scan is in progress. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## 5.2.1.11 Operational Amplifier (r_opamp)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_OPAMP_Open (opamp_ctrl_t *const p_api_ctrl, opamp_cfg_t const *const p_cfg) |

| | |
|---|---|
| fsp_err_t | R_OPAMP_InfoGet (opamp_ctrl_t *const p_api_ctrl, opamp_info_t *const p_info) |
| fsp_err_t | R_OPAMP_Start (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) |
| fsp_err_t | R_OPAMP_Stop (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask) |
| fsp_err_t | R_OPAMP_StatusGet (opamp_ctrl_t *const p_api_ctrl, opamp_status_t *const p_status) |
| fsp_err_t | R_OPAMP_Trim (opamp_ctrl_t *const p_api_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args) |
| fsp_err_t | R_OPAMP_Close (opamp_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface.

# Overview

The OPAMP HAL module provides a high level API for signal amplification applications and supports the OPAMP peripheral available on RA MCUs.

### Features

- Low power or high-speed mode
- Start by software or AGT compare match
- Stop by software or ADC conversion end (stop by ADC conversion end only supported on op-amp channels configured to start by AGT compare match)
- Trimming available on some MCUs (see hardware manual)

# Configuration

### Build Time Configurations for r_opamp

The following build time configurations are defined in fsp_cfg/r_opamp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > Operational Amplifier (r_opamp)

This module can be added to the Stacks tab via New Stack > Analog > Operational Amplifier (r_opamp).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_opamp0 | Module name. |
| AGT Start Trigger Configuration (N/A unless AGT Start Trigger is Selected for the Channel) | • AGT1 Compare Match Starts OPAMPs 0 and 2 if configured for AGT Start, AGT0 Compare Match Starts OPAMPs 1 and 3 if configured for AGT Start<br>• AGT1 Compare Match Starts OPAMPs 0 and 1 if configured for AGT Start, AGT0 Compare Match Starts OPAMPs 2 and 3 if configured for AGT Start<br>• AGT1 Compare Match Starts all OPAMPs configured for AGT Start | AGT1 Compare Match Starts all OPAMPs configured for AGT Start | Configure which AGT channel event triggers which op-amp channel. The AGT compare match event only starts the op-amp channel if the AGT Start trigger is selected in the Trigger configuration for the channel. |
| Power Mode | MCU Specific Options | | Configure the op-amp based on power or speed requirements. This setting affects the minimum required stabilization time. Middle speed is not available for all MCUs. |
| Trigger Channel 0 | MCU Specific Options | | Select the event triggers to start or stop op-amp channel 0. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel. |
| Trigger Channel 1 | MCU Specific Options | | Select the event triggers to start or stop op-amp channel 1. If the event trigger is |

| | | | |
|---|---|---|---|
| | | | selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel. |
| Trigger Channel 2 | • Software Start Software Stop<br>• AGT Start Software Stop<br>• AGT Start ADC Stop | Software Start Software Stop | Select the event triggers to start or stop op-amp channel 2. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel. |
| Trigger Channel 3 | MCU Specific Options | | Select the event triggers to start or stop op-amp channel 3. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel. |
| OPAMP AMP0OS | MCU Specific Options | | Select output to connect to AMP0O pin |
| OPAMP AMP0PS | MCU Specific Options | | Select input to connect to AMP0+ pin |
| OPAMP AMP0MS | MCU Specific Options | | Select input to connect to AMP0- pin |
| OPAMP AMP1PS | MCU Specific Options | | Select input to connect to AMP1+ pin |
| OPAMP AMP1MS | MCU Specific Options | | Select input to connect to AMP1- pin |
| OPAMP AMP2PS | MCU Specific Options | | Select input to connect to AMP2+ pin |
| OPAMP AMP2MS | MCU Specific Options | | Select input to connect to AMP2- pin |

### Clock Configuration

The OPAMP runs on PCLKB.

### Pin Configuration

To use the OPAMP HAL module, the port pins for the channels receiving the analog input must be set as inputs on the **Pins** tab of the RA Configuration editor.

Refer to the most recent FSP Release Notes for any additional operational limitations for this module.

# Usage Notes

### Trimming the OPAMP

- On MCUs that support trimming, the op-amp trim register is set to the factory default after the Open API is called.
- This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default trim values.
- Supported on selected MCUs. See hardware manual for details.
- Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).
- This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling the trim API with the command OPAMP_TRIM_CMD_START again.
    - The trim procedure works as follows:
    - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
    - Connect a fixed voltage to the Pch (+) input.
    - Connect the Nch (-) input to the op-amp output to create a voltage follower.
    - Ensure the op-amp is operating and stabilized.
    - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
    - Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
    - Iterate over the following loop 5 times:
        - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
        - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
        - If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
    - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
    - Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
    - Iterate over the following loop 5 times:
        - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
        - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
        - If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

### Examples

### Basic Example

This is a basic example of minimal use of the R_OPAMP in an application. The example demonstrates configuring OPAMP channel 0 for high speed mode, starting the OPAMP and reading the status of the OPAMP channel running. It also verifies that the stabilization wait time is the expected time for

selected power mode

```
#define OPAMP_EXAMPLE_CHANNEL (0U)
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    assert(FSP_SUCCESS == err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    assert(FSP_SUCCESS == err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
}
```

### Trim Example

This example demonstrates the typical trimming procedure for opamp channel 0 using R_OPAMP_Trim() API.

```
#ifndef OPAMP_EXAMPLE_CHANNEL
 #define OPAMP_EXAMPLE_CHANNEL (0U)
#endif
#ifndef OPAMP_EXAMPLE_ADC_CHANNEL
 #define OPAMP_EXAMPLE_ADC_CHANNEL (ADC_CHANNEL_2)
#endif
#define ADC_SCAN_END_DELAY (100U)
#define OPAMP_TRIM_LOOP_COUNT (5)
#define ADC_SCAN_END_MAX_TIMEOUT (0xFFFF)
uint32_t          g_callback_event_counter = 0;
```

```
opamp_trim_args_t trim_args_ch =
{
    .channel = OPAMP_EXAMPLE_CHANNEL,
    .input   = OPAMP_TRIM_INPUT_PCH
};
/* This callback is called when ADC Scan Complete event is generated. */
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void trimming_example (void)
{
 fsp_err_t err;
 /* On RA2A1, configure negative feedback and put DAC12 signal on AMP0+ Pin. */
    g_opamp_cfg_extend.plus_input_select_opamp0 = OPAMP_PLUS_INPUT_AMPPS7;
    g_opamp_cfg_extend.minus_input_select_opamp0 = OPAMP_MINUS_INPUT_AMPMS7;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    assert(FSP_SUCCESS == err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    assert(FSP_SUCCESS == err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Call trim() for the Pch (+) side input */
    trim_procedure(&trim_args_ch);
    assert(FSP_SUCCESS == err);
    trim_args_ch.input = OPAMP_TRIM_INPUT_NCH;
```

```
 /* Call trim() for the Nch (-) side input */
    trim_procedure(&trim_args_ch);
}
void trim_procedure (opamp_trim_args_t * trim_args)
{
 fsp_err_t err;
 /* Call trim() for the selected channel and input with command OPAMP_TRIM_CMD_START.
*/
    err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_START, trim_args);
    assert(FSP_SUCCESS == err);
 /* Measure the fixed voltage connected to the channel input using the SAR ADC and
save the value
  * (referred to as result_a later in this procedure). */
 /* Reset the ADC callback counter */
    g_callback_event_counter = 0;
    err = R_ADC_ScanStart(&g_adc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Wait for ADC scan complete flag */
    uint32_t timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
    {
       timeout--;
    }
 if (0 == timeout)
    {
       err = FSP_ERR_TIMEOUT;
      assert(FSP_SUCCESS == err);
    }
    uint16_t result_a;
    err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_a);
    assert(FSP_SUCCESS == err);
 /* Iterate over the following loop 5 times: */
 /* Call trim() with command OPAMP_TRIM_CMD_NEXT_STEP for the selected channel and
given input. */
```

```
   uint8_t count = OPAMP_TRIM_LOOP_COUNT;
 while (count > 0)
   {
      count--;
      err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_NEXT_STEP, trim_args);
     assert(FSP_SUCCESS == err);
 /* Reset the ADC callback counter */
     g_callback_event_counter = 0;
 /* Read converted value after trim completes. */
     err = R_ADC_ScanStart(&g_adc_ctrl);
     assert(FSP_SUCCESS == err);
 /* Wait for ADC scan complete flag */
      timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
     {
        timeout--;
     }
 if (0 == timeout)
     {
        err = FSP_ERR_TIMEOUT;
     assert(FSP_SUCCESS == err);
     }
      uint16_t result_b;
      err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_b);
     assert(FSP_SUCCESS == err);
 /* Measure the op-amp output using the SAR ADC (referred to as result_b in the next
step). */
 /* If result_a <= result_b, call trim() for the selected channel and input with
command OPAMP_TRIM_CMD_CLEAR_BIT. */
 if (result_a <= result_b)
     {
           err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_CLEAR_BIT, trim_args);
     assert(FSP_SUCCESS == err);
     }
```

```
    }
}
```

## Data Structures

| | |
|---|---|
| struct | opamp_extended_cfg_t |
| struct | opamp_instance_ctrl_t |

## Macros

| | |
|---|---|
| #define | OPAMP_MASK_CHANNEL_0 |

## Enumerations

| | |
|---|---|
| enum | opamp_trigger_t |
| enum | opamp_agt_link_t |
| enum | opamp_mode_t |
| enum | opamp_plus_input_t |
| enum | opamp_minus_input_t |
| enum | opamp_output_t |

## Variables

| | |
|---|---|
| const opamp_api_t | g_opamp_on_opamp |

## Data Structure Documentation

### ◆ opamp_extended_cfg_t

| struct opamp_extended_cfg_t | | |
|---|---|---|
| OPAMP configuration extension. This extension is required and must be provided in opamp_cfg_t::p_extend. | | |
| Data Fields | | |
| opamp_agt_link_t | agt_link | Configure which AGT links are paired to which channel. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel. |
| opamp_mode_t | mode | Low power, middle speed, or high speed mode. |
| opamp_trigger_t | trigger_channel_0 | Start and stop triggers for |

| | | |
|---|---|---|
| | | channel 0. |
| opamp_trigger_t | trigger_channel_1 | Start and stop triggers for channel 1. |
| opamp_trigger_t | trigger_channel_2 | Start and stop triggers for channel 2. |
| opamp_trigger_t | trigger_channel_3 | Start and stop triggers for channel 3. |
| opamp_plus_input_t | plus_input_select_opamp0 | OPAMP0+ connection. |
| opamp_minus_input_t | minus_input_select_opamp0 | OPAMP0- connection. |
| opamp_output_t | output_select_opamp0 | OPAMP0O connection. |
| opamp_plus_input_t | plus_input_select_opamp1 | OPAMP1+ connection. |
| opamp_minus_input_t | minus_input_select_opamp1 | OPAMP1- connection. |
| opamp_plus_input_t | plus_input_select_opamp2 | OPAMP2+ connection. |
| opamp_minus_input_t | minus_input_select_opamp2 | OPAMP2- connection. |

### ◆ opamp_instance_ctrl_t

| struct opamp_instance_ctrl_t |
|---|
| OPAMP instance control block. DO NOT INITIALIZE. Initialized in opamp_api_t::open(). |

## Macro Definition Documentation

### ◆ OPAMP_MASK_CHANNEL_0

| #define OPAMP_MASK_CHANNEL_0 |
|---|
| Version of code that implements the API defined in this file |

## Enumeration Type Documentation

### ◆ opamp_trigger_t

| enum opamp_trigger_t | |
|---|---|
| Start and stop trigger for the op-amp. | |
| Enumerator | |
| OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP | Start and stop with APIs. |
| OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP | Start by AGT compare match and stop with API. |
| OPAMP_TRIGGER_AGT_START_ADC_STOP | Start by AGT compare match and stop after ADC conversion. |

### ◆ opamp_agt_link_t

| enum opamp_agt_link_t | |
|---|---|
| Which AGT timer starts the op-amp. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel. If OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP is selected for a channel, then no AGT compare match event will start that op-amp channel. | |
| Enumerator | |
| OPAMP_AGT_LINK_AGT1_OPAMP_0_2_AGT0_OPAMP_1_3 | OPAMP channel 0 and 2 are started by AGT1 compare match. OPAMP channel 1 and 3 are started by AGT0 compare match. |
| OPAMP_AGT_LINK_AGT1_OPAMP_0_1_AGT0_OPAMP_2_3 | OPAMP channel 0 and 1 are started by AGT1 compare match. OPAMP channel 2 and 3 are started by AGT0 compare match. |
| OPAMP_AGT_LINK_AGT1_OPAMP_0_1_2_3 | All OPAMP channels are started by AGT1 compare match. |

### ◆ opamp_mode_t

| enum opamp_mode_t | |
|---|---|
| Op-amp mode. | |
| Enumerator | |
| OPAMP_MODE_LOW_POWER | Low power mode. |
| OPAMP_MODE_MIDDLE_SPEED | Middle speed mode (not supported on all MCUs) |
| OPAMP_MODE_HIGH_SPEED | High speed mode. |

### ◆ opamp_plus_input_t

| enum opamp_plus_input_t | |
|---|---|
| Options to connect AMPnPS pins. | |
| Enumerator | |
| OPAMP_PLUS_INPUT_NONE | No Connection. |
| OPAMP_PLUS_INPUT_AMPPS0 | Set AMPPS0. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS1 | Set AMPPS1. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS2 | Set AMPPS2. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS3 | Set AMPPS3. See hardware manual for channel specific options. |
| OPAMP_PLUS_INPUT_AMPPS7 | Set AMPPS7. See hardware manual for channel specific options. |

### ◆ opamp_minus_input_t

| enum opamp_minus_input_t | |
|---|---|
| Options to connect AMPnMS pins. | |
| Enumerator | |
| OPAMP_MINUS_INPUT_NONE | No Connection. |
| OPAMP_MINUS_INPUT_AMPMS0 | Set AMPMS0. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS1 | Set AMPMS1. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS2 | Set AMPMS2. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS3 | Set AMPMS3. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS4 | Set AMPMS4. See hardware manual for channel specific options. |
| OPAMP_MINUS_INPUT_AMPMS7 | Set AMPMS7. See hardware manual for channel specific options. |

### ◆ opamp_output_t

| enum opamp_output_t | |
|---|---|
| Options to connect AMP0OS pin. | |
| Enumerator | |
| OPAMP_OUTPUT_NONE | No Connection. |
| OPAMP_OUTPUT_AMPOS0 | Set AMPOS0. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS1 | Set AMPOS1. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS2 | Set AMPOS2. See hardware manual for channel specific options. |
| OPAMP_OUTPUT_AMPOS3 | Set AMPOS3. See hardware manual for channel specific options. |

## Function Documentation

### ◆ R_OPAMP_Open()

fsp_err_t R_OPAMP_Open ( opamp_ctrl_t *const *p_api_ctrl*, opamp_cfg_t const *const *p_cfg* )

Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements opamp_api_t::open.

The op-amp is not operational until the opamp_api_t::start is called. If the op-amp is configured to start after AGT compare match, the op-amp is not operational until opamp_api_t::start and the associated AGT compare match event occurs.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Initialize the OPAMP module. */
   err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block is already opened. |
| FSP_ERR_INVALID_ARGUMENT | An attempt to configure OPAMP in middle speed mode on MCU that does not support middle speed mode. |

### ◆ R_OPAMP_InfoGet()

fsp_err_t R_OPAMP_InfoGet ( opamp_ctrl_t *const *p_api_ctrl*, opamp_info_t *const *p_info* )

Provides the minimum stabilization wait time in microseconds. Implements opamp_api_t::infoGet.

- Example:

```
/* Look up the required stabilization wait time. */
opamp_info_t info;
   err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
   assert(FSP_SUCCESS == err);
/* Wait for the OPAMP to stabilize. */
R_BSP_SoftwareDelay(info.min_stabilization_wait_us,
BSP_DELAY_UNITS_MICROSECONDS);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | information on opamp_power_mode stored in p_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Start()

fsp_err_t R_OPAMP_Start ( opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask* )

If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements opamp_api_t::start.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Start the OPAMP module. */
   err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Op-amp started or hardware triggers enabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_ARGUMENT | channel_mask includes a channel that does not exist on this MCU. |

### ◆ R_OPAMP_Stop()

fsp_err_t R_OPAMP_Stop ( opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask* )

Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements opamp_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Op-amp stopped or hardware triggers disabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_ARGUMENT | channel_mask includes a channel that does not exist on this MCU. |

### ◆ R_OPAMP_StatusGet()

fsp_err_t R_OPAMP_StatusGet ( opamp_ctrl_t *const *p_api_ctrl*, opamp_status_t *const *p_status* )

Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed. Implements opamp_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operating status of each op-amp provided in p_status. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Trim()

fsp_err_t R_OPAMP_Trim ( opamp_ctrl_t *const *p_api_ctrl*, opamp_trim_cmd_t const *cmd*, opamp_trim_args_t const *const *p_args* )

On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Not supported on all MCUs. See hardware manual for details. Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling trim() with command OPAMP_TRIM_CMD_START again.

Implements opamp_api_t::trim.

See "User Offset Trimming" in the OPAMP section of the relevant hardware manual. The trim procedure works as follows:

- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Connect a fixed voltage to the Pch (+) input.
- Connect the Nch (-) input to the op-amp output to create a voltage follower.
- Ensure the op-amp is operating and stabilized.
- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
    - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
    - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
    - If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
- Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
    - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.

- Measure the op-amp output using the SAR ADC (referred to as B in the next step).
- If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_UNSUPPORTED | Trimming is not supported on this MCU. |
| FSP_ERR_INVALID_STATE | The command is not valid in the current state of the trim state machine. |
| FSP_ERR_INVALID_ARGUMENT | The requested channel is not operating or the trim procedure is not in progress for this channel/input combination. |
| FSP_ERR_INVALID_MODE | Trim is not allowed in low power mode. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_OPAMP_Close()

fsp_err_t R_OPAMP_Close ( opamp_ctrl_t *const *p_api_ctrl*)

Stops the op-amps. Implements opamp_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## Variable Documentation

### ◆ g_opamp_on_opamp

const opamp_api_t g_opamp_on_opamp

OPAMP Implementation of OPAMP interface.

## 5.2.1.12 SDADC Channel Configuration (r_sdadc)

Modules » Analog

### Functions

fsp_err_t R_SDADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)

| | |
|---|---|
| fsp_err_t | R_SDADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend) |
| fsp_err_t | R_SDADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_SDADC_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id) |
| fsp_err_t | R_SDADC_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_SDADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_SDADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_SDADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| fsp_err_t | R_SDADC_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend) |
| fsp_err_t | R_SDADC_Close (adc_ctrl_t *p_ctrl) |

### Detailed Description

Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The SDADC module supports the following features:

- 24 bit maximum resolution
- Configure scans to include:
  - Multiple analog channels
  - Outputs of OPAMP0 (P side) and OPAMP1 (N side) of SDADC channel 4
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
- Configurable scan mode:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
- Supports averaging converted samples
- Optional callback when single conversion, entire scan, or calibration completes
- Supports reading converted data
- Sample and hold support

### Selecting an ADC

All RA MCUs have an ADC (r_adc). Only select RA MCUs have an SDADC. When selecting between them, consider these factors. Refer to the hardware manual for details.

| | ADC | SDADC |
|---|---|---|
| Availability | Available on all RA MCUs. | Available on select RA MCUs. |
| Resolution | The ADC has a maximum resolution of 12, 14, or 16 bits depending on the MCU. | The SDADC has a maximum accuracy of 24 bits. |
| Number of Channels | The ADC has more channels than the SDADC. | The SDADC 5 channels, one of which is tied to OPAMP0 and OPAMP1. |
| Frequency | The ADC sampling time is shorter (more samples per second). | The SDADC sampling time is longer (fewer samples per second). |
| Settling Time | The ADC does not have a settling time when switching between channels. | The SDADC requires a settling time when switching between channels. |

# Configuration

### Build Time Configurations for r_sdadc

The following build time configurations are defined in fsp_cfg/r_sdadc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Analog > ADC (r_sdadc)

This module can be added to the Stacks tab via New Stack > Analog > ADC (r_sdadc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_adc0 | Module name. |
| Mode | • Single Scan<br>• Continuous Scan | Continuous Scan | In single scan mode, all channels are converted once per start trigger, and conversion stops after all enabled channels are scanned. In continuous scan mode, conversion starts after a start |

| | | | |
|---|---|---|---|
| | | | trigger, then continues until stopped in software. |
| Resolution | • 16 Bit<br>• 24 Bit | 24 Bit | Select 24-bit or 16-bit resolution. |
| Alignment | • Right<br>• Left | Right | Select left or right alignment. |
| Trigger | MCU Specific Options | | Select conversion start trigger. Conversion can be started in software, or conversion can be started when a hardware event occurs if the hardware event is linked to the SDADC peripheral using the ELC API. |
| Vref Source | • Internal<br>• External | Internal | Vref can be source internally and output on the SBIAS pin, or Vref can be input from VREFI. |
| Vref Voltage | • 0.8 V<br>• 1.0 V<br>• 1.2 V<br>• 1.4 V<br>• 1.6 V<br>• 1.8 V<br>• 2.0 V<br>• 2.2 V<br>• 2.4 V | 1.0 V | Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%. |
| Callback | Name must be a valid C symbol | NULL | Enter the name of the callback function to be called when conversion completes or a scan ends. |
| Conversion End Interrupt Priority | MCU Specific Options | | [Required] Select the interrupt priority for the conversion end interrupt. |
| Scan End Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the scan end interrupt. |
| Calibration End Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the calibration end interrupt. |

**Configurations for Analog > SDADC Channel Configuration (r_sdadc)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Input | • Differential<br>• Single Ended | Differential | Select differential or single-ended input. |
| Stage 1 Gain | • 1<br>• 2<br>• 3<br>• 4<br>• 8 | 1 | Select the gain for stage 1 of the PGA. Must be 1 for single-ended input. |
| Stage 2 Gain | • 1<br>• 2<br>• 4<br>• 8 | 1 | Select the gain for stage 2 of the PGA. Must be 1 for single-ended input. |
| Oversampling Ratio | • 64<br>• 128<br>• 256<br>• 512<br>• 1024<br>• 2048 | 256 | Select the oversampling ratio for the PGA. Must be 256 for single-ended input. |
| Polarity (Valid for Single-Ended Input Only) | • Positive<br>• Negative | Positive | Select positive or negative polarity for single-ended input. VBIAS (1.0 V typical) is connected on the opposite input. |
| Conversions to Average per Result | • Do Not Average (Interrupt after Each Conversion)<br>• Average 8<br>• Average 16<br>• Average 32<br>• Average 64 | Do Not Average (Interrupt after Each Conversion) | Select the number of conversions to average for each result. The ADC_EVENT_CONVERSION_END event occurs after each average, or after each individual conversion if averaging is disabled. |
| Invert (Valid for Negative Single-Ended Input Only) | • Result Not Inverted<br>• Result Inverted | Result Not Inverted | Select whether to invert negative single-ended input. When the result is inverted, the lowest measurable voltage gives a result of 0, and the highest measurable voltage gives a result of 2^resolution - 1. |
| Number of Conversions Per Scan | Refer to the RA Configuration tool for available options. | 1 | Number of conversions on this channel before AUTOSCAN moves to the next channel. When all conversions of all channels are complete, the |

ADC_EVENT_SCAN_END event occurs.

### Clock Configuration

The SDADC clock clock is configurable on the clocks tab.

The SDADC clock must be 4 MHz when the SDADC is used.

### Pin Configuration

The ANSDnP (n = 0-3) pins are analog input channels that can be used with the SDADC.

# Usage Notes

### Scan Procedure

In this document, the term "scan" refers to the AUTOSCAN feature of the SDADC, which works as follows:

1. Conversions are performed on enabled channels in ascending order of channel number. All conversions required for a single channel are completed before the sequencer moves to the next channel.
2. Conversions are performed at the rate (in Hz) of the SDADC oversampling clock frequency / oversampling ratio (configured per channel). FSP uses the normal mode SDADC oversampling clock frequency.
3. If averaging is enabled for the channel, the number of conversions to average are performed before each conversion end interrupt occurs.

4. If the number of conversions for the channel is more than 1, SDADC performs the number of conversions requested. These are performed consecutively. There is a settling time associated with switching channels. Performing all of the requested conversions for each channel at a time avoids this settling time after the first conversion.

   If averaging is enabled for the channel, each averaged result counts as a single conversion.

5. Continues to the next enabled channel only after completing all conversions requested.
6. After all enabled channels are scanned, a scan end interrupt occurs. The driver supports single-scan and continuous scan operation modes.
   - Single-scan mode performs one scan per trigger (hardware trigger or software start using R_SDADC_ScanStart).
   - In continuous scan mode, the scan is restarted after each scan completes. A single trigger is required to start continuous operation of the SDADC.

### When Interrupts Are Not Enabled

If interrupts are not enabled, the R_SDADC_StatusGet() API can be used to poll the SDADC to determine when the scan has completed. The R_SDADC_Read() API function is used to access the converted SDADC result. This applies to both normal scans and calibration scans.

### Calibration

Calibration is required to use the SDADC if any channel is configured for differential mode. Call R_SDADC_Calibrate() after open, and prior to any other function, then wait for a calibration complete

event before using the SDADC. R_SDADC_Calibrate() should not be called if all channels are configured for single-ended mode.

# Examples

### Basic Example

This is a basic example of minimal use of the SDADC in an application.

```
void sdadc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Calibrate all differential channels. */
 sdadc_calibrate_args_t calibrate_args;
    calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
    calibrate_args.channel = ADC_CHANNEL_0;
    err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
    assert(FSP_SUCCESS == err);
 /* Wait for calibration to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_ScanStart(). */
    (void) R_SDADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
```

```
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint32_t channel1_conversion_result;
 R_SDADC_Read32(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
}
```

### Using DTC or DMAC with the SDADC

If desired, the DTC or DMAC can be used to store each conversion result in a circular buffer. An example configuration is below.

```
/* Example DTC transfer settings to used with SDADC. */
/* The transfer length should match the total number of conversions per scan. This
example assumes the SDADC is
 * configured to scan channel 1 three times, then channel 2 and channel 4 once, for a
total of 5 conversions. */
#define SDADC_EXAMPLE_TRANSFER_LENGTH (5)
uint32_t g_sdadc_example_buffer[SDADC_EXAMPLE_TRANSFER_LENGTH];
transfer_info_t g_sdadc_transfer_info DTC_TRANSFER_INFO_ALIGNMENT =
{
    .transfer_settings_word_b.dest_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED,
    .transfer_settings_word_b.repeat_area    = TRANSFER_REPEAT_AREA_DESTINATION,
    .transfer_settings_word_b.irq            = TRANSFER_IRQ_END,
    .transfer_settings_word_b.chain_mode     = TRANSFER_CHAIN_MODE_DISABLED,
    .transfer_settings_word_b.src_addr_mode = TRANSFER_ADDR_MODE_FIXED,
    .transfer_settings_word_b.mode           = TRANSFER_MODE_REPEAT,
 /* NOTE: The data transferred will contain a 24-bit converted value in bits 23:0.
Bit 24 contains a status flag
  * indicating if the result overflowed or not. Bits 27:25 contain the channel number
+ 1. The settings for
  * resolution and alignment and ignored when DTC or DMAC is used. */
    .transfer_settings_word_b.size           = TRANSFER_SIZE_4_BYTE,
 /* NOTE: It is strongly recommended to enable averaging on all channels or no
channels when using DTC with SDADC
```

```
  * because the result register is different when averaging is used. If averaging is
enabled on all channels,
  * set transfer_info_t::p_src to &R_SDADC->ADAR. */
    .p_src = (void const *) &R_SDADC0->ADCR,
    .p_dest = &g_sdadc_example_buffer[0],
    .length = SDADC_EXAMPLE_TRANSFER_LENGTH,
};
void sdadc_dtc_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Calibrate all differential channels. */
 sdadc_calibrate_args_t calibrate_args;
    calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
    calibrate_args.channel = ADC_CHANNEL_0;
    err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
    assert(FSP_SUCCESS == err);
 /* Wait for calibration to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_ScanStart(). */
    (void) R_SDADC_ScanStart(&g_adc0_ctrl);
 /* After each conversion, the converted data is transferred to the next index in
g_sdadc_example_buffer. After
  * the entire scan completes, the index in g_sdadc_example_buffer resets. The data
```

```
in g_sdadc_example_buffer
  * is:
  * - g_sdadc_example_buffer[0] = SDADC channel 1 conversion 0
  * - g_sdadc_example_buffer[1] = SDADC channel 1 conversion 1
  * - g_sdadc_example_buffer[2] = SDADC channel 1 conversion 2
  * - g_sdadc_example_buffer[3] = SDADC channel 2 conversion 0
  * - g_sdadc_example_buffer[4] = SDADC channel 4 conversion 0
  *//* At any point in the application after the first scan completes, the most
recent data for channel 2 can be read
  * from the buffer like this. Shifting removes the unrelated bits in the result
register and propagates the sign
  * bit so the value can be interpreted as a signed result. This assumes channel 2 is
configured in differential
  * mode. */
    int32_t channel_2_data = (int32_t) (g_sdadc_example_buffer[3] << 8) >> 8;
 FSP_PARAMETER_NOT_USED(channel_2_data);
}
```

## Data Structures

| | |
|---|---|
| struct | sdadc_calibrate_args_t |
| struct | sdadc_channel_cfg_t |
| struct | sdadc_scan_cfg_t |
| struct | sdadc_extended_cfg_t |
| struct | sdadc_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sdadc_vref_src_t |
| enum | sdadc_vref_voltage_t |
| enum | sdadc_channel_input_t |
| enum | sdadc_channel_stage_1_gain_t |
| enum | sdadc_channel_stage_2_gain_t |
| enum | sdadc_channel_oversampling_t |

| | |
|---|---|
| enum | sdadc_channel_polarity_t |
| enum | sdadc_channel_average_t |
| enum | sdadc_channel_inversion_t |
| enum | sdadc_channel_count_formula_t |
| enum | sdadc_calibration_t |

## Data Structure Documentation

### ◆ sdadc_calibrate_args_t

| struct sdadc_calibrate_args_t | | |
|---|---|---|
| Structure to pass to the adc_api_t::calibrate p_extend argument. | | |
| Data Fields | | |
| adc_channel_t | channel | Which channel to calibrate. |
| sdadc_calibration_t | mode | Calibration mode. |

### ◆ sdadc_channel_cfg_t

| struct sdadc_channel_cfg_t |
|---|
| SDADC per channel configuration. |

### ◆ sdadc_scan_cfg_t

| struct sdadc_scan_cfg_t | | |
|---|---|---|
| SDADC active channel configuration | | |
| Data Fields | | |
| uint32_t | scan_mask | Channels/bits: bit 0 is ch0; bit 15 is ch15. |

### ◆ sdadc_extended_cfg_t

| struct sdadc_extended_cfg_t | | |
|---|---|---|
| SDADC configuration extension. This extension is required and must be provided in adc_cfg_t::p_extend. | | |
| Data Fields | | |
| uint8_t | conv_end_ipl | Conversion end interrupt priority. |
| IRQn_Type | conv_end_irq | |
| sdadc_vref_src_t | vref_src | Source of Vref (internal or external) |

| | | |
|---|---|---|
| sdadc_vref_voltage_t | vref_voltage | Voltage of Vref, required for both internal and external Vref. If Vref is from an external source, the voltage must match the specified voltage within 3%. |
| sdadc_channel_cfg_t const * | p_channel_cfgs[SDADC_MAX_NUM_CHANNELS] | Configuration for each channel, set to NULL if unused. |

### ◆ sdadc_instance_ctrl_t

| struct sdadc_instance_ctrl_t |
|---|
| ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). |

## Enumeration Type Documentation

### ◆ sdadc_vref_src_t

| enum sdadc_vref_src_t | |
|---|---|
| Source of Vref. | |
| Enumerator | |
| SDADC_VREF_SRC_INTERNAL | Vref is internally sourced, can be output as SBIAS. |
| SDADC_VREF_SRC_EXTERNAL | Vref is externally sourced from the VREFI pin. |

### ◆ sdadc_vref_voltage_t

| enum sdadc_vref_voltage_t | |
|---|---|
| Voltage of Vref. | |
| Enumerator | |
| SDADC_VREF_VOLTAGE_800_MV | Vref is 0.8 V. |
| SDADC_VREF_VOLTAGE_1000_MV | Vref is 1.0 V. |
| SDADC_VREF_VOLTAGE_1200_MV | Vref is 1.2 V. |
| SDADC_VREF_VOLTAGE_1400_MV | Vref is 1.4 V. |
| SDADC_VREF_VOLTAGE_1600_MV | Vref is 1.6 V. |
| SDADC_VREF_VOLTAGE_1800_MV | Vref is 1.8 V. |
| SDADC_VREF_VOLTAGE_2000_MV | Vref is 2.0 V. |
| SDADC_VREF_VOLTAGE_2200_MV | Vref is 2.2 V. |
| SDADC_VREF_VOLTAGE_2400_MV | Vref is 2.4 V (only valid for external Vref) |

### ◆ sdadc_channel_input_t

| enum sdadc_channel_input_t | |
|---|---|
| Per channel input mode. | |
| Enumerator | |
| SDADC_CHANNEL_INPUT_DIFFERENTIAL | Differential input. |
| SDADC_CHANNEL_INPUT_SINGLE_ENDED | Single-ended input. |

### ◆ sdadc_channel_stage_1_gain_t

| enum sdadc_channel_stage_1_gain_t | |
|---|---|
| Per channel stage 1 gain options. | |
| Enumerator | |
| SDADC_CHANNEL_STAGE_1_GAIN_1 | Gain of 1. |
| SDADC_CHANNEL_STAGE_1_GAIN_2 | Gain of 2. |
| SDADC_CHANNEL_STAGE_1_GAIN_3 | Gain of 3 (only valid for stage 1) |
| SDADC_CHANNEL_STAGE_1_GAIN_4 | Gain of 4. |
| SDADC_CHANNEL_STAGE_1_GAIN_8 | Gain of 8. |

### ◆ sdadc_channel_stage_2_gain_t

| enum sdadc_channel_stage_2_gain_t | |
|---|---|
| Per channel stage 2 gain options. | |
| Enumerator | |
| SDADC_CHANNEL_STAGE_2_GAIN_1 | Gain of 1. |
| SDADC_CHANNEL_STAGE_2_GAIN_2 | Gain of 2. |
| SDADC_CHANNEL_STAGE_2_GAIN_4 | Gain of 4. |
| SDADC_CHANNEL_STAGE_2_GAIN_8 | Gain of 8. |

### ◆ sdadc_channel_oversampling_t

| enum sdadc_channel_oversampling_t | |
|---|---|
| Per channel oversampling ratio. | |
| Enumerator | |
| SDADC_CHANNEL_OVERSAMPLING_64 | Oversampling ratio of 64. |
| SDADC_CHANNEL_OVERSAMPLING_128 | Oversampling ratio of 128. |
| SDADC_CHANNEL_OVERSAMPLING_256 | Oversampling ratio of 256. |
| SDADC_CHANNEL_OVERSAMPLING_512 | Oversampling ratio of 512. |
| SDADC_CHANNEL_OVERSAMPLING_1024 | Oversampling ratio of 1024. |
| SDADC_CHANNEL_OVERSAMPLING_2048 | Oversampling ratio of 2048. |

### ◆ sdadc_channel_polarity_t

| enum sdadc_channel_polarity_t | |
|---|---|
| Per channel polarity, valid for single-ended input only. | |
| Enumerator | |
| SDADC_CHANNEL_POLARITY_POSITIVE | Positive-side single-ended input. |
| SDADC_CHANNEL_POLARITY_NEGATIVE | Negative-side single-ended input. |

### ◆ sdadc_channel_average_t

| enum sdadc_channel_average_t | |
|---|---|
| Per channel number of conversions to average before conversion end callback. | |
| Enumerator | |
| SDADC_CHANNEL_AVERAGE_NONE | Do not average (callback for each conversion) |
| SDADC_CHANNEL_AVERAGE_8 | Average 8 samples for each conversion end callback. |
| SDADC_CHANNEL_AVERAGE_16 | Average 16 samples for each conversion end callback. |
| SDADC_CHANNEL_AVERAGE_32 | Average 32 samples for each conversion end callback. |
| SDADC_CHANNEL_AVERAGE_64 | Average 64 samples for each conversion end callback. |

### ◆ sdadc_channel_inversion_t

| enum sdadc_channel_inversion_t | |
|---|---|
| Per channel polarity, valid for negative-side single-ended input only. | |
| Enumerator | |
| SDADC_CHANNEL_INVERSION_OFF | Do not invert conversion result. |
| SDADC_CHANNEL_INVERSION_ON | Invert conversion result. |

### ◆ sdadc_channel_count_formula_t

| enum sdadc_channel_count_formula_t | |
|---|---|
| Select a formula to specify the number of conversions. The following symbols are used in the formulas:<br>• N: Number of conversions<br>• n: sdadc_channel_cfg_t::coefficient_n, do not set to 0 if m is 0<br>• m: sdadc_channel_cfg_t::coefficient_m, do not set to 0 if n is 0<br>Either m or n must be non-zero. | |
| Enumerator | |
| SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL | N = 32 * (2 ^ n - 1) + m * 2 ^ n. |
| SDADC_CHANNEL_COUNT_FORMULA_LINEAR | N = (32 * n) + m. |

### ◆ sdadc_calibration_t

| enum sdadc_calibration_t | |
|---|---|
| Calibration mode. | |
| Enumerator | |
| SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET | Use internal reference to calibrate offset and gain. |
| SDADC_CALIBRATION_EXTERNAL_OFFSET | Use external reference to calibrate offset. |
| SDADC_CALIBRATION_EXTERNAL_GAIN | Use external reference to calibrate gain. |

## Function Documentation

### ◆ R_SDADC_Open()

fsp_err_t R_SDADC_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Applies power to the SDADC and initializes the hardware based on the user configuration. As part of this initialization, the SDADC clock is configured and enabled. If an interrupt priority is non-zero, enables an interrupt which will call a callback to notify the user when a conversion, scan, or calibration is complete. R_SDADC_Calibrate() must be called after this function before using the SDADC if any channels are used in differential mode. Implements adc_api_t::open().

***Note***

*This function delays at least 2 ms as required by the SDADC power on procedure.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Control block is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt is disabled |

### ◆ R_SDADC_ScanCfg()

fsp_err_t R_SDADC_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_extend* )

Configures the enabled channels of the ADC. Pass a pointer to sdadc_scan_cfg_t to p_extend. Implements adc_api_t::scanCfg().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_InfoGet()

fsp_err_t R_SDADC_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

Returns the address of the lowest number configured channel, the total number of results to be read in order to read the results of all configured channels, the size of each result, and the ELC event enumerations. Implements adc_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_ScanStart()

fsp_err_t R_SDADC_ScanStart ( adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, enables hardware triggers. Otherwise, starts a scan. Implements adc_api_t::scanStart().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan started or hardware triggers enabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |

### ◆ R_SDADC_ScanGroupStart()

fsp_err_t R_SDADC_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id* )

adc_api_t::scanStart is not supported on the SDADC. Use scanStart instead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SDADC_ScanStop()

fsp_err_t R_SDADC_ScanStop ( adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, disables hardware triggers. Otherwise, stops any in-progress scan started by software. Implements adc_api_t::scanStop().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Scan stopped or hardware triggers disabled successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_StatusGet()

fsp_err_t R_SDADC_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Returns the status of a scan started by software, including calibration scans. It is not possible to determine the status of a scan started by a hardware trigger. Implements adc_api_t::scanStatusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | No software scan or calibration is in progress. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Read()

fsp_err_t R_SDADC_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. Implements adc_api_t::read().

***Note***

*The result stored in p_data is signed when the SDADC channel is configured in differential mode.*
*Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Read32()

fsp_err_t R_SDADC_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads the most recent conversion result from a channel. Implements adc_api_t::read32().

***Note***

*The result stored in p_data is signed when the SDADC channel is configured in differential mode. When the SDADC is configured for 24-bit resolution and right alignment, the sign bit is bit 23, and the upper 8 bits are 0. When the SDADC is configured for 16-bit resolution and right alignment, the sign bit is bit 15, and the upper 16 bits are 0.*
*Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_OffsetSet()

fsp_err_t R_SDADC_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset* )

Sets the offset. Offset is applied after stage 1 of the input channel. Offset can only be applied when the channel is configured for differential input. Implements adc_api_t::offsetSet().

Note: The offset is cleared if adc_api_t::calibrate() is called. The offset can be re-applied if necessary after the the callback with event ADC_EVENT_CALIBRATION_COMPLETE is called.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | See p_instance_ctrl in adc_api_t::offsetSet(). |
| [in] | reg_id | See reg_id in adc_api_t::offsetSet(). |
| [in] | offset | Must be between -15 and 15, offset (mV) = 10.9376 mV * offset_steps / stage 1 gain. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Offset updated successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Calibrate()

fsp_err_t R_SDADC_Calibrate ( adc_ctrl_t *const *p_ctrl*, void const * *p_extend* )

Requires sdadc_calibrate_args_t passed to p_extend. Calibrates the specified channel. Calibration is not required or supported for single-ended mode. Calibration must be completed for differential mode before using the SDADC. A callback with the event ADC_EVENT_CALIBRATION_COMPLETE is called when calibration completes. Implements adc_api_t::calibrate().

During external offset calibration, apply a differential voltage of 0 to ANSDnP - ANSDnN, where n is the input channel and ANSDnP is OPAMP0 for channel 4 and ANSDnN is OPAMP1 for channel 4. Complete external offset calibration before external gain calibration.

During external gain calibration apply a voltage between 0.4 V / total_gain and 0.8 V / total_gain. The differential voltage applied during calibration is corrected to a conversion result of 0x7FFFFF, which is the maximum possible positive differential measurement.

This function clears the offset value. If offset is required after calibration, it must be reapplied after calibration is complete using adc_api_t::offsetSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calibration began successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_IN_USE | A conversion or calibration is in progress. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SDADC_Close()

fsp_err_t R_SDADC_Close ( adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the SDADC peripheral. Implements adc_api_t::close().

***Note***

*This function delays at least 3 us as required by the SDADC24 stop procedure.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## 5.2.1.13 SDADC_B Channel Configuration (r_sdadc_b)

Modules » Analog

### Functions

| | |
|---|---|
| fsp_err_t | R_SDADC_B_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SDADC_B_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend) |
| fsp_err_t | R_SDADC_B_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info) |
| fsp_err_t | R_SDADC_B_ScanStart (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_B_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id) |
| fsp_err_t | R_SDADC_B_ScanStop (adc_ctrl_t *p_ctrl) |
| fsp_err_t | R_SDADC_B_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status) |
| fsp_err_t | R_SDADC_B_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| fsp_err_t | R_SDADC_B_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| fsp_err_t | R_SDADC_B_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| fsp_err_t | R_SDADC_B_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend) |
| fsp_err_t | R_SDADC_B_Close (adc_ctrl_t *p_ctrl) |

### Detailed Description

Driver for the SDADC_B peripheral on RA MCUs. This module implements the ADC Interface.

# Overview

### Features

The SDADC_B module supports the following features:

- 24 bit maximum resolution
- Configure scans to include:
  - Multiple analog channels
- Configurable sampling mode:
  - 4 kHz sampling mode
  - 8 kHz sampling mode
  - 8 kHz/4 kHz Hybrid sampling mode
- Configurable Cut-off frequency of high-pass filter
- Configurable preamplifier gain for each channel
- Configurable phase adjustment for each channel

- Optional callback when conversion completes or zero-cross is detected
- Supports reading converted data
- Supports positive and negative input voltage

### Selecting an ADC

All RA MCUs have an ADC (r_adc). Only select RA MCUs have an SDADC_B. When selecting between them, consider these factors. Refer to the hardware manual for details.

| | ADC | SDADC_B |
|---|---|---|
| Availability | Available on all RA MCUs. | Available on select RA MCUs. |
| Resolution | The ADC has a maximum resolution of 12, 14, or 16 bits depending on the MCU. | The SDADC_B has a maximum accuracy of 24 bits. |
| Number of Channels | The ADC has 4 channels. | The SDADC_B has up to 7 channels. |
| Frequency | The ADC sampling time is shorter (more samples per second). | The SDADC sampling time is longer (fewer samples per second). |

# Configuration

### Build Time Configurations for r_sdadc_b

The following build time configurations are defined in fsp_cfg/r_sdadc_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Clock Configuration

The SDADC_B clock is configurable on the clocks tab.

The SDADC_B clock must be 12 MHz, 12.8 MHz or 16 MHz when the SDADC_B is used.

### Pin Configuration

The ANINn/ANIPn (n = 0-6) pins are analog input channels that can be used with the SDADC_B.

### Virtual Channel Configuration

- When 8KHz/4KHz Hybrid Mode is enabled, selected Virtual Channels 0 to 3 are converted at an 8KHz rate and corresponding Virtual Channels 4 to 7 are converted at a 4KHz rate.
- In some locations, 4KHz Hybrid Mode conversions may be referred to 'Type 2'. All other conversions may be referred to as 'Type 1'. The SDADC_B driver abstracts 'Type 1' and 'Type 2' and populates the user callback arguments with a mask of converted channel IDs. It is the responsibility of the application to know the configuration of provided channel IDs.

- Configuration of enabled virtual channels corresponding to Sampling mode is subject to below constraints:
  - Analog input n (n = 0 to 3) must be configured for Virtual Channel m (m = 4 to 7) if corresponding Analog input n is configured for Virtual Channel n and Hybrid mode is enabled
  - Analog input 4 to 6 cannot be configured for Virtual Channel 4 to 6 when Hybrid mode is enabled
  - Virtual Channel 7 can only be configured when Hybrid mode is enabled

# Usage Notes

### Cut-off frequency of HPF

Cut-off frequency of HPF (High Pass Filter) is selectable by setting SDADHPFCR.COF bit. Value of cut-off frequency of HPF corresponding to each COF value is as below table:

| COF | 4 kHz sampling mode fos = 1.5 MHz | 4 kHz sampling mode fos = 1.6 MHz | 8 kHz sampling mode fos = 3.0 MHz | 8 kHz sampling mode fos = 3.2 MHz | 8 kHz/4 kHz hybrid sampling mode fos = 3.0 MHz | 8 kHz/4 kHz hybrid sampling mode fos = 3.2 MHz |
|---|---|---|---|---|---|---|
| COF 0 | 0.607 Hz | 0.647 Hz | 1.214 Hz | 1.295 Hz | 1.214 Hz | 1.295 Hz |
| COF 1 | 1.214 Hz | 1.295 Hz | 2.427 Hz | 2.589 Hz | 2.427 Hz | 2.589 Hz |
| COF 2 | 2.427 Hz | 2.589 Hz | 4.855 Hz | 5.179 Hz | 4.855 Hz | 5.179 Hz |
| COF 3 | 4.855 Hz | 5.179 Hz | 9.710 Hz | 10.357 Hz | 1.214 Hz (Type 1) 0.607 Hz (Type 2) | 1.295 Hz (Type 1) 0.647 Hz (Type 2) |

### Channel ID used in driver

Channel ID input to R_SDADC_B_Read and R_SDADC_B_Read32 API and Channel Mask provided to user callback is the virtual channel ID and does not necessarily correspond to a physical input channel number. When Hybrid sampling mode is enabled, reading conversion result for channel 4 to 7 returns the 4 kHz conversion (Type 2) result of corresponding channel 0 to 3.

### Scan Procedure

Operation of 24-bit Sigma-Delta A/D Converter is as below:

1. Conversions are performed on enabled channels in ascending order of channel number after software trigger is started using R_SDADC_B_ScanStart() API.
2. Conversions are performed at the rate (in Hz) of the SDADC_B sampling clock frequency.
3. The user callback function will be invoked after enabled channels are scanned.
4. If zero-cross detection interrupt is enabled, a zero-cross detection interrupt occurs in synchronization with the rising edge of the SDADC conversion end interrupt.

### Triggering ELC Events with SDADC_B

- The interrupt signals SDADC_B can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

*Note*

*When using ELC events to directly read converted data, it is the responsibility of the application to ensure that the stabilization period of 80 conversions has passed since starting the converter by using R_SDADC_B_StatusGet() API.*

### Limitations

- After powering on, an internal setup time is necessary. During this time R_SDADC_B_StatusGet() will return ADC_STATE_CALIBRATION_IN_PROGRESS.
- SDADC_B does not operate in software standby mode. To reduce current comsumption, stop and close the SDADC_B module before entering software standby.
- After stopping the conversion using R_SDADC_B_ScanStop() API, it is necessary to wait at least 1.4us before performing conversion again.

## Examples

### Basic Example

This is a basic example of minimal use of the SDADC_B in an application.

```
void sdadc_b_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_B_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_SDADC_B_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_B_ScanStart(). */
    (void) R_SDADC_B_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_CALIBRATION_IN_PROGRESS;
 while (ADC_STATE_CALIBRATION_IN_PROGRESS == status.state)
    {
 R_SDADC_B_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint32_t channel1_conversion_result;
 R_SDADC_B_Read32(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
```

```
}
```

### Using DTC or DMAC with the SDADC_B

If desired, the DTC or DMAC can be used to store each conversion result in a circular buffer. An example configuration is below.

```
/* Example DTC transfer settings to used with SDADC_B. */
/* The transfer length should match the total number of conversions per scan. This
example assumes the SDADC_B is
 * configured to scan all 7 channels. */
#define SDADC_B_EXAMPLE_TRANSFER_LENGTH (7)
uint32_t g_sdadc_b_example_buffer[SDADC_B_EXAMPLE_TRANSFER_LENGTH];
transfer_info_t g_sdadc_b_transfer_info DTC_TRANSFER_INFO_ALIGNMENT =
{
    .transfer_settings_word_b.dest_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED,
    .transfer_settings_word_b.repeat_area    = TRANSFER_REPEAT_AREA_DESTINATION,
    .transfer_settings_word_b.irq            = TRANSFER_IRQ_END,
    .transfer_settings_word_b.chain_mode     = TRANSFER_CHAIN_MODE_DISABLED,
    .transfer_settings_word_b.src_addr_mode = TRANSFER_ADDR_MODE_FIXED,
    .transfer_settings_word_b.mode           = TRANSFER_MODE_REPEAT,
 /* NOTE: The data transferred will contain a 24-bit converted value in bits 23:0.
The settings for
  * resolution and alignment are ignored when DTC or DMAC is used. */
    .transfer_settings_word_b.size           = TRANSFER_SIZE_4_BYTE,
    .p_src = (void const *) &R_SDADC_B->SDADCR0,
    .p_dest = &g_sdadc_b_example_buffer[0],
    .length = SDADC_B_EXAMPLE_TRANSFER_LENGTH,
};
void sdadc_b_dtc_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_B_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Start a scan by calling R_SDADC_B_ScanStart(). */
    (void) R_SDADC_B_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_CALIBRATION_IN_PROGRESS;
 while (ADC_STATE_CALIBRATION_IN_PROGRESS == status.state)
    {
 R_SDADC_B_StatusGet(&g_adc0_ctrl, &status);
    }
 /* After each conversion, the converted data is transferred to the next index in
g_sdadc_b_example_buffer. After
  * the entire scan completes, the index in g_sdadc_b_example_buffer resets. The data
in g_sdadc_b_example_buffer
  * is:
  * - g_sdadc_b_example_buffer[0] = SDADC_B channel 0
  * - g_sdadc_b_example_buffer[1] = SDADC_B channel 1
  * - g_sdadc_b_example_buffer[2] = SDADC_B channel 2
  * - g_sdadc_b_example_buffer[3] = SDADC_B channel 3
  * - g_sdadc_b_example_buffer[4] = SDADC_B channel 4
  * - g_sdadc_b_example_buffer[5] = SDADC_B channel 5
  * - g_sdadc_b_example_buffer[6] = SDADC_B channel 6
  *//* At any point in the application after the first scan completes, the most
recent data for channel 2 can be read
  * from the buffer like this. Conversion data is signed integer value. */
    int32_t channel_2_data = (int32_t) g_sdadc_b_example_buffer[2];
 FSP_PARAMETER_NOT_USED(channel_2_data);
}
```

## Data Structures

| | |
|---|---|
| struct | sdadc_b_scan_cfg_t |
| struct | sdadc_b_extended_cfg_t |
| struct | sdadc_b_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sdadc_b_oper_clk_t |
| enum | sdadc_b_channel_mode_t |
| enum | sdadc_b_channel_power_t |
| enum | sdadc_b_samp_mode_t |
| enum | sdadc_b_channel_gain_t |
| enum | sdadc_b_channel_hpf_t |
| enum | sdadc_b_resolution_t |
| enum | sdadc_b_zc_channel_t |
| enum | sdadc_b_zc_output_mode_t |
| enum | sdadc_b_zc_falling_edge_detection_t |
| enum | sdadc_b_zc_rising_edge_detection_t |
| enum | sdadc_b_cutoff_t |

## Data Structure Documentation

### ◆ sdadc_b_scan_cfg_t

| struct sdadc_b_scan_cfg_t | | |
|---|---|---|
| SDADC_B active channel configuration. | | |
| Data Fields | | |
| union sdadc_b_scan_cfg_t | __unnamed__ | |
| sdadc_b_cutoff_t | hpf_cutoff | Cut-off frequency of HPF. |
| sdadc_b_channel_gain_t | gain_setting[SDADC_B_MAX_NUM_CHANNELS - 1] | Gain setting. |
| sdadc_b_channel_hpf_t | hpf_setting[SDADC_B_MAX_NUM_CHANNELS] | High pass filter on-off. |
| uint16_t | phase_adjustment[SDADC_B_MAX_NUM_CHANNELS] | Phase adjustment for each channels. |

### ◆ sdadc_b_extended_cfg_t

| struct sdadc_b_extended_cfg_t |
|---|
| SDADC configuration extension. This extension is required and must be provided in adc_cfg_t::p_extend. |
| Data Fields |

| | | |
|---|---|---|
| sdadc_b_oper_clk_t | oper_clk | Operating clock of the digital block. |
| sdadc_b_samp_mode_t | sampling_mode | Sampling mode select. |
| sdadc_b_scan_cfg_t const * | p_channel_cfg | Pointer to original channel config data. |
| uint8_t | conv_end_ipl | Conversion end interrupt priority. |
| IRQn_Type | conv_end_irq | Conversion type 1 end IRQ number. |
| IRQn_Type | conv_end_irq2 | Conversion type 2 end IRQ number. |
| uint8_t | zc_ipl | Zero-cross detection 0 interrupt priority. |
| IRQn_Type | zc_irq | Zero-cross detection 0 IRQ number. |
| uint8_t | zc_ipl2 | Zero-cross detection 1 interrupt priority. |
| IRQn_Type | zc_irq2 | Zero-cross detection 1 IRQ number. |
| union sdadc_b_extended_cfg_t | __unnamed__ | |

### ◆ sdadc_b_instance_ctrl_t

| struct sdadc_b_instance_ctrl_t | | |
|---|---|---|
| SDADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open(). | | |
| Data Fields | | |
| adc_cfg_t const * | p_cfg | |
| uint32_t | opened | Boolean to verify that the Unit has been initialized. |
| uint32_t | channel_mask | Channel mask to keep track of channels enabled in scanCfg. |
| struct sdadc_b_instance_ctrl_t | results | |
| uint32_t | setup_time_cnt | Period of 80 conversions for internal setup time. |
| uint8_t | calibration_complete | Calibration is completed if set. |

## Enumeration Type Documentation

### ◆ sdadc_b_oper_clk_t

| enum sdadc_b_oper_clk_t | |
|---|---|
| Operating clock of the digital block. | |
| Enumerator | |
| SDADC_B_CLOCK_DISABLE | Disable operating clock. |
| SDADC_B_CLOCK_IS_12MHZ | Clock frequency is 12MHz or 12.8MHz. |
| SDADC_B_CLOCK_IS_16MHZ | Clock frequency is 16MHz. |

### ◆ sdadc_b_channel_mode_t

| enum sdadc_b_channel_mode_t | |
|---|---|
| Per channel operation mode. | |
| Enumerator | |
| SDADC_B_ELECTRIC_CHARGE_RESET | Electric charge reset. |
| SDADC_B_NORMAL_OPERATION | Normal operation. |

### ◆ sdadc_b_channel_power_t

| enum sdadc_b_channel_power_t | |
|---|---|
| Per channel power-on control. | |
| Enumerator | |
| SDADC_B_CHANNEL_POWER_OFF | Power off. |
| SDADC_B_CHANNEL_POWER_ON | Power on. |

### ◆ sdadc_b_samp_mode_t

| enum sdadc_b_samp_mode_t | |
|---|---|
| Sampling mode select. | |
| Enumerator | |
| SDADC_B_4KHZ_SAMPLING_MODE | 4 kHz sampling mode |
| SDADC_B_8KHZ_SAMPLING_MODE | 8 kHz sampling mode |
| SDADC_B_HYBRID_SAMPLING_MODE | 8 kHz / 4 kHz hybird sampling mode |

### ◆ sdadc_b_channel_gain_t

| enum sdadc_b_channel_gain_t | |
|---|---|
| Per channel preamplifier gain options. | |
| Enumerator | |
| SDADC_B_CHANNEL_GAIN_1 | Gain of 1. |
| SDADC_B_CHANNEL_GAIN_2 | Gain of 2. |
| SDADC_B_CHANNEL_GAIN_4 | Gain of 4. |
| SDADC_B_CHANNEL_GAIN_8 | Gain of 8. |
| SDADC_B_CHANNEL_GAIN_16 | Gain of 16. |
| SDADC_B_CHANNEL_GAIN_32 | Gain of 32. |

### ◆ sdadc_b_channel_hpf_t

| enum sdadc_b_channel_hpf_t | |
|---|---|
| Per channel HPF bypass. | |
| Enumerator | |
| SDADC_B_CHANNEL_HPF_ENABLE | HPF enable. |
| SDADC_B_CHANNEL_HPF_DISABLE | HPF disable. |

### ◆ sdadc_b_resolution_t

| enum sdadc_b_resolution_t | |
|---|---|
| SDADC data resolution definitions | |
| Enumerator | |
| SDADC_B_RESOLUTION_24_BIT | 24 bit resolution |
| SDADC_B_RESOLUTION_16_BIT | 16 bit resolution |

### ◆ sdadc_b_zc_channel_t

| enum sdadc_b_zc_channel_t | |
|---|---|
| Zero-cross detection channel | |
| Enumerator | |
| SDADC_B_ZC_CHANNEL_2_OR_3 | Detect channel 2 (ZCCTL0) or channel 3 (ZCCTL1) |
| SDADC_B_ZC_CHANNEL_1_OR_0 | Detect channel 1 (ZCCTL0) or channel 0 (ZCCTL1) |

### ◆ sdadc_b_zc_output_mode_t

| enum sdadc_b_zc_output_mode_t | |
|---|---|
| Zero-cross detection output mode. | |
| Enumerator | |
| SDADC_B_ZC_PULSE_OUTPUT_MODE | Pulse output mode. |
| SDADC_B_ZC_LEVEL_OUTPUT_MODE | Level output mode. |

### ◆ sdadc_b_zc_falling_edge_detection_t

| enum sdadc_b_zc_falling_edge_detection_t | |
|---|---|
| Zero-cross detection output mode. | |
| Enumerator | |
| SDADC_B_ZC_FALLING_EDGE_DETECTION_DISABLE | Disabled. |
| SDADC_B_ZC_FALLING_EDGE_DETECTION_ENABLE | Enabled. |

### ◆ sdadc_b_zc_rising_edge_detection_t

| enum sdadc_b_zc_rising_edge_detection_t | |
|---|---|
| Zero-cross detection output mode. | |
| Enumerator | |
| SDADC_B_ZC_RISING_EDGE_DETECTION_DISABLE | Disabled. |
| SDADC_B_ZC_RISING_EDGE_DETECTION_ENABLE | Enabled. |

### ◆ sdadc_b_cutoff_t

| enum sdadc_b_cutoff_t | |
|---|---|
| HPF cut off. The enum value is to set to SDADHPFCR register. See table "Cut-off frequency of HPF" in the SDADC section of the relevant hardware manual | |
| Enumerator | |
| SDADC_B_CUTOFF_00B | Cut-off frequency 0. |
| SDADC_B_CUTOFF_01B | Cut-off frequency 1. |
| SDADC_B_CUTOFF_10B | Cut-off frequency 2. |
| SDADC_B_CUTOFF_11B | Cut-off frequency 3. |

## Function Documentation

### ◆ R_SDADC_B_Open()

fsp_err_t R_SDADC_B_Open ( adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg* )

Applies power to the SDADC_B and initializes the hardware based on the user configuration. Enabling interrupts which will call a callback to notify the user when a conversion is completed or a zero-cross is detected. Implements adc_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Control block is already open. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel configuration |

### ◆ R_SDADC_B_ScanCfg()

fsp_err_t R_SDADC_B_ScanCfg ( adc_ctrl_t * *p_ctrl*, void const *const *p_extend* )

Configures the enabled channels of the ADC. Implements adc_api_t::scanCfg().

***Note***

*This function is not compatible with Hybrid Mode operation.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_adc_info. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_MODE | Hybrid mode channel configuration is invalid |
| FSP_ERR_INVALID_CHANNEL | Invalid channel configuration |
| FSP_ERR_INVALID_STATE | Converter operation must be stopped before reconfiguring |

### ◆ R_SDADC_B_InfoGet()

fsp_err_t R_SDADC_B_InfoGet ( adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info* )

Returns the address of the lowest number configured channel, the total number of results to be read in order to read the results of all configured channels, the size of each result, and the ELC event enumerations. Implements adc_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion is started successfully. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |

### ◆ R_SDADC_B_ScanStart()

fsp_err_t R_SDADC_B_ScanStart ( adc_ctrl_t * *p_ctrl*)

adc_api_t::scanStart().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion is started successfully. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |

### ◆ R_SDADC_B_ScanGroupStart()

fsp_err_t R_SDADC_B_ScanGroupStart ( adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id* )

adc_api_t::scanGroupStart is not supported on the SDADC_B. Use scanStart instead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SDADC_B_ScanStop()

fsp_err_t R_SDADC_B_ScanStop ( adc_ctrl_t * *p_ctrl*)

adc_api_t::scanStop().

***Note***

*According to Hardware specification, after stopping the conversion, it is necessary to wait at least 1.4us before performing conversion again.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion is started successfully. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_ASSERTION | An input pointer is NULL or an input parameter is invalid. |

### ◆ R_SDADC_B_StatusGet()

fsp_err_t R_SDADC_B_StatusGet ( adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status* )

Returns the status of a scan including calibration scans. Implements adc_api_t::scanStatusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module status stored in the provided pointer p_status |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ R_SDADC_B_Read()

fsp_err_t R_SDADC_B_Read ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data* )

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. When the SDADC_B is configured for 16-bit resolution, the sign bit is bit 15 and the upper 16 bits are 0. Implements adc_api_t::read().

***Note***

*The result stored in p_data is signed.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_DATA | Result buffer has not been updated with valid data. |

### ◆ R_SDADC_B_Read32()

fsp_err_t R_SDADC_B_Read32 ( adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data* )

Reads the most recent conversion result from a channel. Implements adc_api_t::read32().

***Note***

*The result stored in p_data is signed.*
*When the SDADC is configured for 24-bit resolution, the upper 8 bits are sign extended.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Conversion result in p_data. |
| FSP_ERR_ASSERTION | An input pointer was NULL or an input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_INVALID_DATA | Result buffer has not been updated with valid data. |

### ◆ R_SDADC_B_OffsetSet()

fsp_err_t R_SDADC_B_OffsetSet ( adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset* )

adc_api_t::offsetSet is not supported on the SDADC_B.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SDADC_B_Calibrate()

fsp_err_t R_SDADC_B_Calibrate ( adc_ctrl_t *const *p_ctrl*, void const * *p_extend* )

Calibration is performed automatically each time a scan is started. adc_api_t::calibrate is not supported on the SDADC_B.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SDADC_B_Close()

fsp_err_t R_SDADC_B_Close ( adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the SDADC_B peripheral. Implements adc_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Instance control block closed successfully. |
| FSP_ERR_ASSERTION | An input pointer was NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

## 5.2.2 AI

Modules

### Detailed Description

Artificial Intelligence Modules.

**Modules**

| | |
|---|---|
| | Ethos-U (rm_ethosu)<br>Middleware to support Ethos-U NPU driver. This module implements the Ethos-U Interface. |
| | Reality AI Data Collector (rm_rai_data_collector)<br>Middleware to implement the Data Collector for Reality AI applications. This module implements the Data Collector Interface. |
| | Reality AI Data Shipper (rm_rai_data_shipper)<br>Middleware to implement the Data Shipper for Reality AI applications. This module implements the Data Shipper Interface. |

### 5.2.2.1 Ethos-U (rm_ethosu)

Modules » AI

**Functions**

| | |
|---|---|
| fsp_err_t | RM_ETHOSU_Open (rm_ethosu_ctrl_t *p_api_ctrl, rm_ethosu_cfg_t const *const p_cfg) |
| fsp_err_t | RM_ETHOSU_CallbackSet (rm_ethosu_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_ethosu_callback_args_t *), void *const p_context, rm_ethosu_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_ETHOSU_Close (rm_ethosu_ctrl_t *p_api_ctrl) |

**Detailed Description**

Middleware to support Ethos-U NPU driver. This module implements the Ethos-U Interface.

# Overview

RM_ETHOSU module is a wrapper of the Ethos-U core driver.

**Features**

- Initialization Ethos-U core driver
- NPU secure and privilege level can be configured
- Callback Events
    - NPU finishes inference job

# Configuration

### Build Time Configurations for rm_ethosu

The following build time configurations are defined in fsp_cfg/rm_ethosu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Selects if code for parameter checking is to be included in the build. |
| Debug log level | • RM_ETHOSU_LOG_LEVEL_OFF<br>• RM_ETHOSU_LOG_LEVEL_ERROR<br>• RM_ETHOSU_LOG_LEVEL_WARNING<br>• RM_ETHOSU_LOG_LEVEL_INFO<br>• RM_ETHOSU_LOG_LEVEL_DEBUG | RM_ETHOSU_LOG_LEVEL_OFF | Defines ethosu log level. |

### Configurations for AI > Arm Ethos-U Core Driver Wrapper (rm_ethosu)

This module can be added to the Stacks tab via New Stack > AI > Arm Ethos-U Core Driver Wrapper (rm_ethosu).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_ethosu0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function that will be called from the interrupt service routine (ISR). |
| NPU Interrupt Priority | MCU Specific Options | | Select the NPU interrupt priority. |
| Enable Secure Mode | • Disabled<br>• Enabled | Enabled | Set Ethos-U NPU secure mode. |
| Enable Privilege Mode | • Disabled<br>• Enabled | Enabled | Set Ethos-U NPU privilege mode. |

### Clock Configuration

The NPU clock is configurable on the clocks tab.

# Usage Notes

Here are the steps to create an AI application in e2studio. Please refer to the example code for details.

### Getting Started: Creating an AI Project with NPU stacks

Start by creating a new C++ project in e² studio. On the Stacks tab, add New Stack > AI > TFLM Core Lib. On the BSP tab, stack and heap need to be configured. On the Clocks tab, NPU clock needs to be configured. Generate the project content. Please note an Ethos-U core driver instance will be automatically created. It will be used by RM_ETHOSU API to initialize/deinitialize Ethos-U driver.The application must also use the same instance to call other Ethos-U core driver API if needed.

### Getting Started: Confirming operators used in the model

Users can use tools like Netron to find out what operators are used by the model (.tflite). If there are operators not yet supported by Ethos-U55, then they need to be run on MCU. There are two types of kernels: optimized kernel based on CMSIS-NN, and reference kernel. Optimized kernel (CMSIS-NN kernel) should always be preferred on ARM MCUs. Please refer to https://github.com/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/docs/optimized_kernel_implementations.md for more information.

### Getting Started: Learning inference process framework

Users can refer to inference process https://gitlab.arm.com/artificial-intelligence/ethos-u/ethos-u-core-software/-/tree/main/applications for how to integrate TFLM inference functions, e.g parsing model, creating interpreter, allocating tensors and running inference. Please note by default lots of operators are registered to micro_op_resolver in micro_mutable_all_ops_resolver.h. What operators are needed should be determined by your model.

### Getting Started: Defining functions and PMU events for MCU performance profiling

ticks_per_second() and GetCurrentTimeTicks() need to be defined for MCU profiling. Here is an example

```
namespace tflite {
    uint32_t ticks_per_second();
    uint32_t GetCurrentTimeTicks();
    uint32_t ticks_per_second() { return US_IN_SECOND; }
    uint32_t GetCurrentTimeTicks() {
       static bool is_initialized = false;
       if (!is_initialized) {
            ARM_PMU_Enable();
            DCB->DEMCR |= DCB_DEMCR_TRCENA_Msk;
            ARM_PMU_CYCCNT_Reset();
            ARM_PMU_CNTR_Enable(PMU_CNTENSET_CCNTR_ENABLE_Msk);
            is_initialized = true;
```

```
        }
        return ARM_PMU_Get_CCNTR();
    }
}
```

There are 4 PMU event counters. Each can be used to monitor an event, e.g

```
#define ETHOSU_PMU_EVENT_0 ETHOSU_PMU_CYCLE
#define ETHOSU_PMU_EVENT_1 ETHOSU_PMU_NPU_ACTIVE
```

### Getting Started: Overriding weak functions in application

There are weak functions that need to be overridden if necessary.

| Module | Function | Description |
|---|---|---|
| NPU Driver | ethosu_flush_dcache | Flush/clean the data cache by address and size. Passing NULL as p argument expects the whole cache to be flushed. |
| NPU Driver | ethosu_invalidate_dcache | Invalidate the data cache by address and size. Passing NULL as p argument expects the whole cache to be invalidated. |
| NPU Driver | ethosu_mutex_create | define mutex API based on RTOS |
| NPU Driver | ethosu_mutex_destroy | define mutex API based on RTOS |
| NPU Driver | ethosu_mutex_lock | define mutex API based on RTOS |
| NPU Driver | ethosu_mutex_unlock | define mutex API based on RTOS |
| NPU Driver | ethosu_semaphore_create | define semaphore API based on RTOS |
| NPU Driver | ethosu_semaphore_destroy | define semaphore API based on RTOS |
| NPU Driver | ethosu_semaphore_take | define semaphore API based on RTOS |
| NPU Driver | ethosu_semaphore_give | define semaphore API based on RTOS |
| NPU Core | ethosu_inference_begin | Callback invoked just before the |

| | | |
|---|---|---|
| | | inference is started. |
| NPU Core | ethosu_inference_end | Callback invoked just after the inference has completed. |

### Getting Started: Creating AI inference

Create inference job and run, e.g

```
InferenceJob job(std::string(modelName),
                DataPtr(networkModelData, networkModelDataLen),
                {DataPtr(inputData, inputDataLen)},
                {DataPtr(outputData, outputDataLen)},
                {DataPtr(expectedOutputData, expectedOutputDataLen)},
                numBytesToPrint);
// Run job
bool failed = inferenceProcess.runJob(job);
```

### Limitations

Users have to implement weak functions if needed. Please refer to https://git.mlplatform.org/ml/ethos-u/ethos-u-core-driver.git/tree/README.md for further information.

Users have to update linker script if they want to save AI model and data to external memories.

## Examples

### Basic Example

This is a basic example of Ethos-U NPU application.

```
/* These variables are auto generated. */
extern struct ethosu_driver g_ethosu_dev;
extern rm_ethosu_cfg_t g_ethosu_cfg;
extern rm_ethosu_instance_ctrl_t g_ethosu_ctrl;
extern "C"
{
 /* Override the SysTick Handler. */
 extern void SysTick_Handler(void);
 void SysTick_Handler(void)
    {
 // Disable systick, preventing new systick interrupt to fire while we call the Ethos-
```

```
U monitor
      SysTick->CTRL = 0;
 if (ethosuDrv != NULL) {
           ethosuMonitor.monitorSample(ethosuDrv);
 // Restart the systick timer
           SysTick_Config(usDelay);
      }
   }
 void ethosu_inference_begin(struct ethosu_driver *drv, void *)
   {
      ethosuDrv = drv;
      ethosuMonitor.configure(drv, pmuEventConfig);
      SysTick_Config(usDelay);
   }
 void ethosu_inference_end(struct ethosu_driver *drv, void *)
   {
 // Disable polling
      SysTick->CTRL = 0;
      ethosuDrv = NULL;
      ethosuMonitor.monitorSample(drv);
      ethosuMonitor.release(drv);
   }
}
void rm_ethosu_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
   err = RM_ETHOSU_Open(&g_ethosu_ctrl, &g_ethosu_cfg);
 if (FSP_SUCCESS != err)
   {
 /* Handle any errors. */
   }
 /* Initialize event recorder */
   EventRecorderInitialize(EventRecordAll, 1);
   InferenceJob job(std::string(modelName),
```

```
                DataPtr(networkModelData, networkModelDataLen),
                {DataPtr(inputData, inputDataLen)},
                {DataPtr(outputData, outputDataLen)},
                {DataPtr(expectedOutputData, expectedOutputDataLen)},
                numBytesToPrint);
 // Run job
 bool failed = inferenceProcess.runJob(job);
    printf("Status of executed job: %s\n", failed ? "Failed\n" : "Success\n");
    err = RM_ETHOSU_Close(&g_ethosu_ctrl);
 if (FSP_SUCCESS != err)
    {
 /* Handle any errors. */
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_ethosu_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ethosu_instance_ctrl_t

| struct rm_ethosu_instance_ctrl_t |
|---|
| Instance control block. This is private to the FSP and should not be used or modified by the application. |

## Function Documentation

### ◆ RM_ETHOSU_Open()

fsp_err_t RM_ETHOSU_Open ( rm_ethosu_ctrl_t * *p_api_ctrl*, rm_ethosu_cfg_t const *const *p_cfg* )

Enable Ethos-U NPU and initialize driver. Implements rm_ethosu_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables NPU module
- Initializes Ethos-U NPU driver.
- Provides the handle for use with other API functions.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to the control structure. |
| p_cfg | Pointer to a configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | NPU initialized successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |
| FSP_ERR_INVALID_ARGUMENT | Failed to initialize ethosu driver |

### ◆ RM_ETHOSU_CallbackSet()

fsp_err_t RM_ETHOSU_CallbackSet ( rm_ethosu_ctrl_t *const *p_api_ctrl*, void(*)(rm_ethosu_callback_args_t *) *p_callback*, void *const *p_context*, rm_ethosu_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements rm_ethosu_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ RM_ETHOSU_Close()

| fsp_err_t RM_ETHOSU_Close ( rm_ethosu_ctrl_t * *p_api_ctrl*) |
|---|
| Disable NPU and set the instance as not open. Implements rm_ethosu_api_t::close. |

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to an opened instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

## 5.2.2.2 Reality AI Data Collector (rm_rai_data_collector)

Modules » AI

### Functions

| | |
|---|---|
| fsp_err_t | RM_RAI_DATA_COLLECTOR_Open (rai_data_collector_ctrl_t *const p_api_ctrl, rai_data_collector_cfg_t const *const p_cfg) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister (rai_data_collector_ctrl_t *const p_api_ctrl, uint8_t channel, void const *p_src) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_BufferReset (rai_data_collector_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_BufferRelease (rai_data_collector_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_ChannelBufferGet (rai_data_collector_ctrl_t *const p_api_ctrl, uint8_t channel, void **pp_buf) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_ChannelWrite (rai_data_collector_ctrl_t *const p_api_ctrl, uint8_t channel, const void *p_buf, uint32_t len) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_SnapshotStart (rai_data_collector_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_RAI_DATA_COLLECTOR_SnapshotStop (rai_data_collector_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_RAI_DATA_COLLECTOR_Close (rai_data_collector_ctrl_t *const p_api_ctrl) |

### Detailed Description

Middleware to implement the Data Collector for Reality AI applications. This module implements the Data Collector Interface.

# Overview

Data Collector is to abstract the collection of data from sensors so that data samples are accumulated into fixed length frames before being made available to application. Support of "snapshot" mode and "data feed" mode are required to accommodate for background and cooperative data collection. Each mode supports 8 sensor channels maximally. Each sensor channel will be captured into a separate frame buffer. Frame buffers shall have the same amount of data samples, however, data type can be different(int32_t, float, uint8_t etc). Users have to make sure that frame buffers will be filled up at the same rate.

When all frame buffers are filled up, they will be provided to the application via data ready callback. After they are consumed, application has to release them by calling RM_RAI_DATA_COLLECTOR_BufferRelease(). For seamless operation, PING-PONG buffer will be used. Ideally buffers will be released before the other set of buffers are filled up. However, it is possible that frame buffers will overrun due to the fact that application may take longer time to process the data in some cases. If it happens, application will be notified with a buffer-overrun event via the error callback. No intervention is required from user side in this case. Buffer overrun will disappear when frame buffers are released. However, if not all sensor channels are configured to work at the same pace, application will get a buffer-out-of-sync error. Users have to reconfigure sensor channels and ensure all of them will work at the same pace. If sensor channels can't work at the same rate, then multiple data collector instances are required.

### Features

- Snapshot mode and data feed mode are supported
- Maximally 8 sensors are supported for each mode
- Mix mode is supported (both snapshot mode and data feed mode are enabled)

### Snapshot Mode

Snapshot mode will periodically pull data from the user-specified places and save to designated frame buffers. It requires a DTC module and a timer module. DTC will work in chain mode, which enables data collection from various, potentially non-linear and different-sized sources. Timer, either General PWM Timer (GPT) or Asynchronous General Purpose Timer, provides activation source for DTC to work periodically. To select this activation source, relevant interrupt has to be configured in the timer module. Application has to start the timer to enable DTC after sensor source addresses are registered.

### Data Feed Mode

Data feed mode will require data producer to push data directly to the designated frame buffer whenever data is ready. Data can be pushed synchronously or asynchronously. Synchronous mode is for use cases that the data producer has a short amount of data to be copied to the frame buffer. Asynchronous mode is to use DTC/DMAC for data transfer. Application has to add DTC/DMAC

modules and initialize transfer descriptors for asynchronous transfer.

**Usage Notes**

1. Do not add DTC stack and timer stack to data collector instance for data feed mode.
2. Users must take care to make sure all channels of a data collector instance will work at the same pace.
3. Interrupt priority level of the timer stack in the data collector instance must be higher (numerically lower) than that of RM COMMS USB PCDC GPT Overflow interrupt.

# Configuration

### Build Time Configurations for rm_rai_data_collector

The following build time configurations are defined in fsp_cfg/rm_rai_data_collector_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Max Number Of Channels | Value must be a positive integer less than or equal to 16 | 16 | Max number of channels. |

### Configurations for AI > Data Collector (rm_rai_data_collector)

This module can be added to the Stacks tab via New Stack > AI > Data Collector (rm_rai_data_collector).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_rai_data_collector0 | Module name. |
| ID | Value must be a positive integer. | 0 | Instance ID |
| Frame Buffer Length | Value must be a positive integer greater than 0 | 100 | Length of frame buffers in data samples. |
| Data Ready Callback | Name must be a valid C symbol | rai_data_collector0_callback | Callback function on data ready. |
| Error Callback | Name must be a valid C symbol | rai_data_collector0_error_callback | Callback function for error events. |
| Data Feed Mode | | | |
| Data Feed Mode > Channel 0 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch0 | Channel name |

| | | | |
|---|---|---|---|
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 1 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch1 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 2 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch2 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 3 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch3 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 4 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch4 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 5 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch5 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 6 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch6 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Data Feed Mode > Channel 7 | | | |
| Name | Must be valid C variable name | dc0_data_feed_ch7 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Channels | Value must be an integer between 0 and 8 | 0 | Number of Data Feed Mode channels. |
| Snapshot Mode | | | |
| Snapshot Mode > Channel 0 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch0 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 1 | | | |

| | | | |
|---|---|---|---|
| Name | Must be valid C variable name | dc0_snapshot_ch1 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 2 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch2 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 3 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch3 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 4 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch4 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 5 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch5 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 6 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch6 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Snapshot Mode > Channel 7 | | | |
| Name | Must be valid C variable name | dc0_snapshot_ch7 | Channel name |
| Data Type | MCU Specific Options | | Channel Data Type |
| Channels | Value must be an integer between 0 and 8 | 0 | Number of snapshot mode channels. |
| DTC Transfer Count | Value must be a positive integer greater than 0 | 1 | DTC transfer count on each activation |

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

# Examples

### Snapshot Mode Example

```
extern rai_data_collector_cfg_t g_dc_cfg;
extern rai_data_collector_instance_t g_dc_ctrl;
static rai_data_collector_frame_buffer_t g_frame_buf[DC_TEST_CHAN_COUNT];
static uint8_t g_sensor_buf[DC_TEST_CHAN_COUNT][DC_TEST_CHAN_BUFFER_LEN];
static bool g_data_ready = false;
static bool g_exit = false;
static rai_data_collector_error_event_t g_dc_error_event =
RAI_DATA_COLLECTOR_ERROR_TYPE_NONE;
static rai_data_collector_callback_args_t g_data_ready_callback_arg;
/* Data ready callback called when all sensor data are ready. */
void data_ready_callback_test (const rai_data_collector_callback_args_t * p_args)
{
 /* Time consuming job should be done in the application. Just save a copy here. */
    g_data_ready_callback_arg.frame_buf_len = p_args->frame_buf_len;
    g_data_ready_callback_arg.frames       = p_args->frames;
    g_data_ready_callback_arg.instance_id   = p_args->instance_id;
 for (uint8_t i = 0; i < p_args->frames; i++)
    {
       g_frame_buf[i] = p_args->p_frame_buf[i];
    }
    g_data_ready_callback_arg.p_frame_buf = g_frame_buf;
    g_data_ready = true;
}
/* Error callback when there is buffer overflow. */
void error_callback_test (const rai_data_collector_error_callback_args_t * p_args)
{
    g_dc_error_event |= p_args->event;
}
void rm_rai_data_collector_snapshot_mode_example ()
{
 fsp_err_t err;
    err = RM_RAI_DATA_COLLECTOR_Open(&g_dc_ctrl, &g_dc_cfg);
```

```
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
for (uint8_t i = 0; i < DC_TEST_CHAN_COUNT; i++)
   {
RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister(&g_dc_ctrl, i, g_sensor_buf[i]);
     assert(FSP_SUCCESS == err);
   }
   err = RM_RAI_DATA_COLLECTOR_SnapshotStart(&g_dc_ctrl);
   assert(FSP_SUCCESS == err);
do
   {
/* Process error event */
if (RAI_DATA_COLLECTOR_ERROR_TYPE_BUF_OUT_OF_SYNC & g_dc_error_event)
     {
/* Error! Please reconfigure sensors to make sure they all work at the same pace.*/
           g_dc_error_event = RAI_DATA_COLLECTOR_ERROR_TYPE_NONE;
     }
if (g_data_ready)
     {
/* Process collected data saved in g_frame_buf. */
           do_data_process(g_data_ready_callback_arg.instance_id,
                           g_data_ready_callback_arg.frames,
                           g_data_ready_callback_arg.frame_buf_len,
                           g_data_ready_callback_arg.p_frame_buf);
           g_data_ready = false;
/* Release buffer when done */
RM_RAI_DATA_COLLECTOR_BufferRelease(&g_dc_ctrl);
     }
   } while (!g_exit);
   err = RM_RAI_DATA_COLLECTOR_SnapshotStop(&g_dc_ctrl);
   assert(FSP_SUCCESS == err);
   err = RM_RAI_DATA_COLLECTOR_Close(&g_dc_ctrl);
   assert(FSP_SUCCESS == err);
}
```

### Data Feed Synchronous Mode Example

*Note*

*The data collector callback functions from the first example apply to this example as well.*

```
extern const timer_cfg_t g_timer1_cfg;
extern gpt_instance_ctrl_t g_timer1_ctrl;
static uint8_t
g_data_feed_sync_mode_buf[DC_TEST_CHAN_COUNT][DC_TEST_CHAN_BUFFER_LEN];
void timer_callback (const timer_callback_args_t * p_args)
{
 fsp_err_t err;
 FSP_PARAMETER_NOT_USED(p_args);
 for (uint8_t i = 0; i < DC_TEST_CHAN_COUNT; i++)
    {
       err = RM_RAI_DATA_COLLECTOR_ChannelWrite(&g_dc_ctrl,
                                                i,
                                                (void *)
g_data_feed_sync_mode_buf[i],
                                                DC_TEST_CHAN_BUFFER_LEN);
      assert(FSP_SUCCESS == err);
    }
}
void rm_rai_data_collector_data_feed_mode_synchronous_example ()
{
 fsp_err_t err;
    err = RM_RAI_DATA_COLLECTOR_Open(&g_dc_ctrl, &g_dc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 R_GPT_Open(&g_timer1_ctrl, &g_timer1_cfg);
 R_GPT_Reset(&g_timer1_ctrl);
 R_GPT_Start(&g_timer1_ctrl);
 do
    {
```

```
 /* Process error event */
 if (RAI_DATA_COLLECTOR_ERROR_TYPE_BUF_OUT_OF_SYNC & g_dc_error_event)
      {
 /* Error! Please reconfigure sensors to make sure they all work at the same pace.*/
            g_dc_error_event = RAI_DATA_COLLECTOR_ERROR_TYPE_NONE;
      }
 if (g_data_ready)
      {
 /* Process collected data saved in g_frame_buf */
            do_data_process(g_data_ready_callback_arg.instance_id,
                            g_data_ready_callback_arg.frames,
                            g_data_ready_callback_arg.frame_buf_len,
                            g_data_ready_callback_arg.p_frame_buf);
            g_data_ready = false;
 /* Release buffer when done */
 RM_RAI_DATA_COLLECTOR_BufferRelease(&g_dc_ctrl);
      }
    } while (!g_exit);
 R_GPT_Stop(&g_timer1_ctrl);
 R_GPT_Close(&g_timer1_ctrl);
    err = RM_RAI_DATA_COLLECTOR_Close(&g_dc_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Data Feed Asynchronous Mode Example

*Note*

*The data collector callback functions from the first example apply to this example as well.*

```
extern const transfer_cfg_t g_dmac_cfg[DC_TEST_CHAN_COUNT];
extern dmac_instance_ctrl_t g_dmac_ctrl[DC_TEST_CHAN_COUNT];
static uint16_t g_data_feed_async_mode_cnt[DC_TEST_CHAN_COUNT];
static uint8_t * g_data_feed_async_mode_buf[DC_TEST_CHAN_COUNT];
void dma_callback (const dmac_callback_args_t * p_args)
{
```

```
    uint8_t chan = *(uint8_t *) p_args->p_context;
    g_data_feed_async_mode_cnt[chan]++;
    if (DC_TEST_REQUIRED_FRAME_LEN == g_data_feed_async_mode_cnt[chan])
    {
 fsp_err_t err =
 RM_RAI_DATA_COLLECTOR_ChannelBufferGet(&g_dc_ctrl, chan, (void **)
&g_data_feed_async_mode_buf[chan]);
      assert(FSP_SUCCESS == err);
       g_data_feed_async_mode_cnt[chan] = 0;
    }
    g_dmac_cfg[chan].p_info->p_dest = g_data_feed_async_mode_buf[chan] +
g_data_feed_async_mode_cnt[chan];
    g_dmac_cfg[chan].p_info->length = 1;
 R_DMAC_Reconfigure(&g_dmac_ctrl[chan], g_dmac_cfg[chan].p_info);
}
void rm_rai_data_collector_data_feed_mode_asynchronous_example ()
{
 fsp_err_t err;
    err = RM_RAI_DATA_COLLECTOR_Open(&g_dc_ctrl, &g_dc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 for (uint8_t i = 0; i < DC_TEST_CHAN_COUNT; i++)
    {
 RM_RAI_DATA_COLLECTOR_ChannelBufferGet(&g_dc_ctrl, i, (void **)
&g_data_feed_async_mode_buf[i]);
      assert(FSP_SUCCESS == err);
       g_dmac_cfg[i].p_info->p_dest = g_data_feed_async_mode_buf[i];
       g_dmac_cfg[i].p_info->p_src = g_sensor_buf[i];
       g_dmac_cfg[i].p_info->length = 1;
 R_DMAC_Open(&g_dmac_ctrl[i], &g_dmac_cfg[i]);
 R_DMAC_Enable(&g_dmac_ctrl[i]);
    }
 do
    {
```

```
/* Process error event */
if (RAI_DATA_COLLECTOR_ERROR_TYPE_BUF_OUT_OF_SYNC & g_dc_error_event)
     {
/* Error! Please reconfigure sensors to make sure they all work at the same pace.*/
           g_dc_error_event = RAI_DATA_COLLECTOR_ERROR_TYPE_NONE;
     }
if (g_data_ready)
     {
/* Process collected data saved in g_frame_buf */
           do_data_process(g_data_ready_callback_arg.instance_id,
                           g_data_ready_callback_arg.frames,
                           g_data_ready_callback_arg.frame_buf_len,
                           g_data_ready_callback_arg.p_frame_buf);
           g_data_ready = false;
/* Release buffer when done */
RM_RAI_DATA_COLLECTOR_BufferRelease(&g_dc_ctrl);
     }
   } while (!g_exit);
for (uint8_t i = 0; i < DC_TEST_CHAN_COUNT; i++)
   {
R_DMAC_Disable(&g_dmac_ctrl[i]);
R_DMAC_Close(&g_dmac_ctrl[i]);
   }
   err = RM_RAI_DATA_COLLECTOR_Close(&g_dc_ctrl);
   assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | rai_data_collector_instance_ctrl_t |

## Data Structure Documentation

### ◆ rai_data_collector_instance_ctrl_t

| struct rai_data_collector_instance_ctrl_t |
|---|
| RAI_DATA_COLLECTOR instance control block. Initialization occurs when RM_RAI_DATA_COLLECTOR_Open() is called. |

## Function Documentation

### ◆ RM_RAI_DATA_COLLECTOR_Open()

fsp_err_t RM_RAI_DATA_COLLECTOR_Open ( rai_data_collector_ctrl_t *const *p_api_ctrl*, rai_data_collector_cfg_t const *const *p_cfg* )

Opens and configures the Data Collector module.

Implements rai_data_collector_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data Collector successfully configured. |
| FSP_ERR_ALREADY_OPEN | Module already open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL or callback is NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister()

fsp_err_t RM_RAI_DATA_COLLECTOR_SnapshotChannelRegister ( rai_data_collector_ctrl_t *const *p_api_ctrl*, uint8_t *channel*, void const * *p_src* )

Config transfer src address for snapshot mode channel

Implements rai_data_collector_api_t::snapshotChannelRegister().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Src addresses are set. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_COLLECTOR_BufferReset()

fsp_err_t RM_RAI_DATA_COLLECTOR_BufferReset ( rai_data_collector_ctrl_t *const *p_api_ctrl*)

Reset to discard accumulated data and start with PING buffer.

***Note***

*Application must stop data transfer on all channels first.*

Implements rai_data_collector_api_t::bufferReset().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data Collector module internal buffers reset. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_COLLECTOR_BufferRelease()

fsp_err_t RM_RAI_DATA_COLLECTOR_BufferRelease ( rai_data_collector_ctrl_t *const *p_api_ctrl*)

Release frame buffer

Implements rai_data_collector_api_t::bufferRelease().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Buffer released. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_COLLECTOR_ChannelBufferGet()

fsp_err_t RM_RAI_DATA_COLLECTOR_ChannelBufferGet ( rai_data_collector_ctrl_t *const *p_api_ctrl*, uint8_t *channel*, void ** *pp_buf* )

Get channel destination buffer address for asynchronous data transfer.

Implements rai_data_collector_api_t::channelBufferGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Buffer available. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_COLLECTOR_ChannelWrite()

fsp_err_t RM_RAI_DATA_COLLECTOR_ChannelWrite ( rai_data_collector_ctrl_t *const *p_api_ctrl*, uint8_t *channel*, const void * *p_buf*, uint32_t *len* )

Synchronouse data transfer using CPU copy.

Implements rai_data_collector_api_t::channelWrite().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data copy completed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_COLLECTOR_SnapshotStart()

fsp_err_t RM_RAI_DATA_COLLECTOR_SnapshotStart ( rai_data_collector_ctrl_t *const *p_api_ctrl*)

Starts snapshot mode channels

Implements rai_data_collector_api_t::snapshotStart().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snapshot mode started. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |
| FSP_ERR_UNSUPPORTED | No snapshot mode channel |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_RAI_DATA_COLLECTOR_SnapshotStop()

fsp_err_t RM_RAI_DATA_COLLECTOR_SnapshotStop ( rai_data_collector_ctrl_t *const *p_api_ctrl*)

Stops snapshot mode channels

Implements rai_data_collector_api_t::snapshotStop().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snapshot mode stopped. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |
| FSP_ERR_UNSUPPORTED | No snapshot mode channel |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_RAI_DATA_COLLECTOR_Close()

fsp_err_t RM_RAI_DATA_COLLECTOR_Close ( rai_data_collector_ctrl_t *const *p_api_ctrl*)

Closes Data Collector module instance.

Implements rai_data_collector_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data Collector module closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.2.3 Reality AI Data Shipper (rm_rai_data_shipper)

Modules » AI

**Functions**

| | |
|---|---|
| fsp_err_t | RM_RAI_DATA_SHIPPER_Open (rai_data_shipper_ctrl_t *const p_api_ctrl, rai_data_shipper_cfg_t const *const p_cfg) |
| fsp_err_t | RM_RAI_DATA_SHIPPER_Read (rai_data_shipper_ctrl_t *const p_api_ctrl, void *const p_buf, uint32_t *const buf_len) |

| | |
|---|---|
| fsp_err_t | RM_RAI_DATA_SHIPPER_Write (rai_data_shipper_ctrl_t *const p_api_ctrl, rai_data_shipper_write_params_t const *p_write_params) |
| fsp_err_t | RM_RAI_DATA_SHIPPER_Close (rai_data_shipper_ctrl_t *const p_api_ctrl) |

### Detailed Description

Middleware to implement the Data Shipper for Reality AI applications. This module implements the Data Shipper Interface.

# Overview

RAI Data Shipper is mainly for sending data collected by RAI Data Collector to PC so that they can be used for Reality AI model training. It utilizes the Communications Middleware Interface and all communications are fully asynchronous. The data being transported may be any combination of the following:

- Sensor data
- System events/errors
- Debug data and diagnostic information, including RAI runtime output

Sensor data is provided by Data Collector via data ready callback. System events and debug data are filled up by application. A callback is used to notify application that communication is finished. RM_COMMS_EVENT_ERROR will be returned if there is any error during data transmission. Please note sensor data buffer must be released back to data collector when data is no longer used by application. If data shipper is the only data consumer, then RM_RAI_DATA_COLLECTOR_BufferRelease must be called on write error, or in the data shipper callback.

*Note*

*Debug data, diagnostic information, and associated events are provided for debug purposes only and are not tracked by upstream Reality AI tools.*

### Features

1. The RAI Data Shipper module supports up to 8 Data Collector instances.
2. The RAI Data Shipper module supports the following interface:
    - RM_COMMS_UART with/without CRC-8
    - RM_COMMS_USB_PCDC with/without CRC-8

# Configuration

### Build Time Configurations for rm_rai_data_shipper

The following build time configurations are defined in fsp_cfg/rm_rai_data_shipper_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

| | | | |
|---|---|---|---|
| Max Number Of DC Instances | Value must be a positive integer between 1 and 8 | 8 | Max number of DC instances to be sent. |

**Configurations for AI > Data Shipper (rm_rai_data_shipper)**

This module can be added to the Stacks tab via New Stack > AI > Data Shipper (rm_rai_data_shipper).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rai_data_shipper0 | Module name |
| Frame Rate Divider | Value must be non-negative | 0 | Skip write requests |
| Callback | Name must be a valid C symbol | rai_data_shipper0_callback | A user callback function on data sent or error. |

**Clock Configuration**

This module has no required clock configurations.

**Pin Configuration**

This module does not use I/O pins.

**Limitations**

1. 2-way communication hasn't been supported yet - data is always sent from device to host.
2. Users shall take care to avoid race condition in the case of multiple data collector instances.
3. To use RM COMMS USB PCDC with Full Speed mode and DMA, frame buffer length needs to be multiple of 2 when 8-bit channel is used.
4. To use RM COMMS USB PCDC with High Speed mode and DMA, frame buffer length needs to be multiple of 2 when 16-bit channel is used. If 8-bit channel is used, its frame buffer length needs to be a multiple of 4.
5. RAI Data Collector (snapshot mode) and RM COMMS USB PCDC must use different timers. They are independent of each other.

# Examples

**Basic Example**

This is a basic example of minimal use of the Data Shipper implementation in an application.

```
void RM_RAI_DATA_SHIPPER_example();

void data_shipper_callback(rai_data_shipper_callback_args_t * p_args);

extern rai_data_shipper_cfg_t g_ds_cfg;

extern rai_data_shipper_instance_ctrl_t g_ds_ctrl;

extern rai_data_collector_instance_ctrl_t g_dc_ctrl;
```

```
static bool g_exit = false;
extern bool g_data_ready;
extern rai_data_collector_callback_args_t g_data_ready_callback_arg;
void RM_RAI_DATA_SHIPPER_example ()
{
 fsp_err_t err;
    err = RM_RAI_DATA_SHIPPER_Open(&g_ds_ctrl, &g_ds_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!g_exit)
    {
 if (g_data_ready)
      {
 rai_data_shipper_write_params_t g_write_params;
            g_write_params.p_sensor_data = &g_data_ready_callback_arg;
 /* Application is responsible to initialize diagnostic information and events. */
 /* Application has no diagnostic information/events to report. */
            g_write_params.diagnostic_data_len = 0;
            g_write_params.events             = 0;
            g_write_params.p_diagnostic_data   = NULL;
            err = RM_RAI_DATA_SHIPPER_Write(&g_ds_ctrl, &g_write_params);
 if (FSP_SUCCESS != err)
      {
 /* Please note callback won't be called on write error */
 /* Release data buffer only when data is no longer used by application. */
 RM_RAI_DATA_COLLECTOR_BufferRelease(&g_dc_ctrl);
            }
            g_data_ready = false;
       }
    }
    err = RM_RAI_DATA_SHIPPER_Close(&g_ds_ctrl);
    assert(FSP_SUCCESS == err);
}
/* Called when all sensor data are sent or there is an error */
```

```
void data_shipper_callback (rai_data_shipper_callback_args_t * p_args)
{
 if (RM_COMMS_EVENT_ERROR == p_args->result)
    {
 /* Communication error */
    }
 /* Release data buffer only when data is no longer used by application. */
 RM_RAI_DATA_COLLECTOR_BufferRelease(&g_dc_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | rai_data_shipper_instance_ctrl_t |

## Data Structure Documentation

### ◆ rai_data_shipper_instance_ctrl_t

| struct rai_data_shipper_instance_ctrl_t |
|---|
| RAI_DATA_SHIPPER instance control block. Initialization occurs when RM_RAI_DATA_SHIPPER_Open() is called. |

## Function Documentation

### ◆ RM_RAI_DATA_SHIPPER_Open()

fsp_err_t RM_RAI_DATA_SHIPPER_Open ( rai_data_shipper_ctrl_t *const *p_api_ctrl*, rai_data_shipper_cfg_t const *const *p_cfg* )

Opens and configures the Data Shipper module.

Implements rai_data_shipper_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data Shipper successfully configured. |
| FSP_ERR_ALREADY_OPEN | Module already open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL or callback is NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_RAI_DATA_SHIPPER_Read()

fsp_err_t RM_RAI_DATA_SHIPPER_Read ( rai_data_shipper_ctrl_t *const *p_api_ctrl*, void *const *p_buf*, uint32_t *const *buf_len* )

Read data.

Implements rai_data_shipper_api_t::read().

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Data Shipper module read not supported |

### ◆ RM_RAI_DATA_SHIPPER_Write()

fsp_err_t RM_RAI_DATA_SHIPPER_Write ( rai_data_shipper_ctrl_t *const *p_api_ctrl*, rai_data_shipper_write_params_t const * *p_write_params* )

Write data. Note this function may be called in ISR.

Implements rai_data_shipper_api_t::write().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Tx buf list created and transmission starts, or write request skipped. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

### ◆ RM_RAI_DATA_SHIPPER_Close()

fsp_err_t RM_RAI_DATA_SHIPPER_Close ( rai_data_shipper_ctrl_t *const *p_api_ctrl*)

Closes Data Shipper module instance.

Implements rai_data_shipper_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data Shipper module closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.3 Audio

Modules

### Detailed Description

Audio Modules.

### Modules

| | |
|---|---|
| | ADPCM Decoder (rm_adpcm_decoder)<br>Middleware to implement the ADPCM Audio Decoder. This module implements the ADPCM Decoder Interface. |
| | Audio Playback PWM (rm_audio_playback_pwm)<br>Driver for the Audio Playback middleware on RA MCUs. This module implements the AUDIO PLAYBACK Interface. |
| | Minimp3 Library<br>Minimalistic, single-header library for decoding MP3. |
| | Pulse Density Modulation Interface (r_pdm)<br>Driver for the PDM peripheral on RA MCUs. This module implements the PDM Interface. |

## 5.2.3.1 ADPCM Decoder (rm_adpcm_decoder)

Modules » Audio

### Functions

| | |
|---|---|
| fsp_err_t | RM_ADPCM_DECODER_Open (adpcm_decoder_ctrl_t *p_ctrl, adpcm_decoder_cfg_t const *const p_cfg) |
| fsp_err_t | RM_ADPCM_DECODER_Decode (adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes) |
| fsp_err_t | RM_ADPCM_DECODER_Reset (adpcm_decoder_ctrl_t *p_ctrl) |
| fsp_err_t | RM_ADPCM_DECODER_Close (adpcm_decoder_ctrl_t *p_ctrl) |

### Detailed Description

Middleware to implement the ADPCM Audio Decoder. This module implements the ADPCM Decoder Interface.

# Overview

### Features

The ADPCM Audio Decoder has the following key features:

- Decodes 4-bit ADPCM input to 16-bit PCM output

# Configuration

### Build Time Configurations for rm_adpcm_decoder

The following build time configurations are defined in fsp_cfg/rm_adpcm_decoder_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Audio > ADPCM Decoder (rm_adpcm_decoder)

This module can be added to the Stacks tab via New Stack > Audio > ADPCM Decoder (rm_adpcm_decoder).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_adpcm_decoder0 | Module name. |

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

# Examples

### Basic Example

This is a basic example of minimal use of the ADPCM Audio Decoder implementation in an application.

```
void rm_adpcm_decoder_example ()
{
```

```
 /* Open the ADPCM audio decoder instance. */
 fsp_err_t err = RM_ADPCM_DECODER_Open(&g_adpcmdec_ctrl, &g_adpcmdec_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Decode the data */
    err = RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl, g_adpcm_stream1, g_pcm_stream,
ADPCM_BUFFER_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Reset the ADPCM audio decoder instance before decoding a new stream. */
    err = RM_ADPCM_DECODER_Reset(&g_adpcmdec_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decode the first chunk of ADPCM data */
    err = RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl, g_adpcm_stream2, g_pcm_stream,
(ADPCM_BUFFER_SIZE_BYTES / 2));
    assert(FSP_SUCCESS == err);
 /* Decode the second chunk of ADPCM data */
    err =
 RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl, &g_adpcm_stream2[ADPCM_BUFFER_SIZE_BYTES /
2], g_pcm_stream,
                            (ADPCM_BUFFER_SIZE_BYTES / 2));
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | adpcm_decoder_instance_ctrl_t |

## Data Structure Documentation

### ◆ adpcm_decoder_instance_ctrl_t

| struct adpcm_decoder_instance_ctrl_t |
|---|
| RM_ADPCM_DECODER instance control block. DO NOT INITIALIZE. Initialized in adpcm_decoder_api_t::open(). |

## Function Documentation

### ◆ RM_ADPCM_DECODER_Open()

fsp_err_t RM_ADPCM_DECODER_Open ( adpcm_decoder_ctrl_t * *p_ctrl*, adpcm_decoder_cfg_t const *const *p_cfg* )

Initializes ADPCM audio decoder device.

Implements adpcm_decoder_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is ready for use. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The instance control structure has already been opened. |

### ◆ RM_ADPCM_DECODER_Decode()

fsp_err_t RM_ADPCM_DECODER_Decode ( adpcm_decoder_ctrl_t *const *p_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t *src_len_bytes* )

Decodes 4bit ADPCM data to 16bit PCM data. It reads ADPCM data from area pointed by inputAddr pointer, decodes the number of samples specified and stores the decoded data in buffer pointed with outputAddr pointer.

Implements adpcm_decoder_api_t::decode().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Decode operation successfully completed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ RM_ADPCM_DECODER_Reset()

fsp_err_t RM_ADPCM_DECODER_Reset ( adpcm_decoder_ctrl_t * *p_ctrl*)

This function resets the ADPCM decoder device.

Implements adpcm_decoder_api_t::reset().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

### ◆ RM_ADPCM_DECODER_Close()

fsp_err_t RM_ADPCM_DECODER_Close ( adpcm_decoder_ctrl_t * *p_ctrl*)

This function closes the ADPCM decoder device.

Implements adpcm_decoder_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module closed. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | Unit is not open. |

## 5.2.3.2 Audio Playback PWM (rm_audio_playback_pwm)

Modules » Audio

### Functions

| | |
|---|---|
| fsp_err_t | RM_AUDIO_PLAYBACK_PWM_Open (audio_playback_ctrl_t *const p_api_ctrl, audio_playback_cfg_t const *const p_cfg) |
| fsp_err_t | RM_AUDIO_PLAYBACK_PWM_Close (audio_playback_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_AUDIO_PLAYBACK_PWM_Start (audio_playback_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_AUDIO_PLAYBACK_PWM_Stop (audio_playback_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_AUDIO_PLAYBACK_PWM_Play (audio_playback_ctrl_t *const p_api_ctrl, void const *const p_buffer, uint32_t length) |

### Detailed Description

Driver for the Audio Playback middleware on RA MCUs. This module implements the AUDIO PLAYBACK Interface.

# Overview

### Features

The Audio Playback with PWM middleware is used to play audio streams at user selected playback rate using Pulse Width Modulation hardware on GPT or AGT timers. This module can play 16-bit or 32-bit(available on selected MCUs) uncompressed, unsigned PCM audio stream when AGT is selected

as PWM interface, and can play 32-bit uncompressed, unsigned PCM audio stream when GPT is used as PWM interface. Note some MCUs have 16-bit GPT timers/channels. In this case audio stream still needs to be 32 bits because the duty cycle register is 32-bit - just the upper 16 bits are ignored. The application code is expected to convert the signed PCM data to unsigned PCM data and scale it with the playback rate before starting the playback.

# Configuration

### Build Time Configurations for rm_audio_playback_pwm

The following build time configurations are defined in fsp_cfg/rm_audio_playback_pwm_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DMAC Support | • Enabled<br>• Disabled | Disabled | Select if DMAC will be used. |

### Configurations for Audio > Audio Playback PWM (rm_audio_playback_pwm)

This module can be added to the Stacks tab via New Stack > Audio > Audio Playback PWM (rm_audio_playback_pwm).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_rm_audio_playback0 | Module name. |
| Playback Speed (Hz) | Must be an integer and greater than 0 | 44100 | Enter playback sample rate in Hz. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | g_rm_audio_playback0_callback | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the playback completes. |
| PWM Output Pin | • Pin A<br>• Pin B | Pin A | Select which timer output pin should be used for audio output. |

### Clock Configuration

The Audio Playback with PWM module does not require a specific clock configuration.

### Pin Configuration

Configure the PWM output pins for selected PWM HAL layer peripheral (AGT/GPT). One of the following pins needs to be selected and enabled as PWM output for selected channel n,

If GPT is used as PWM interface,

- GTIOCAn
- GTIOCBn

If AGT is used as PWM interface,

- AGTOAn
- AGTOBn

## Usage Notes

### DMAC/DTC Integration

DMAC/DTC is used as a lower level transfer instance with this module and is operated in Normal mode to transfer 16 bit or 32 bit data from the audio stream buffer to the PWM peripheral AGT or GPT respectively. Destination address for transfer instance needs to be the Duty Cycle setting register GTCCR for GPT as PWM driver or AGTMA/AGTCMB in case of AGT as PWM driver. The Audio Playback with PWM module internally configures 'Transfer Size' as 2 Bytes if AGT is used for PWM generation, otherwise it configures 'Transfer Size" as 4 Bytes if GPT or AGTW is used for PWM generation. Refer the hardware manual to check whether the MCU supports AGT or the AGTW peripheral.

## Examples

### Basic Example

This is a basic example of minimal use of the RM_AUDIO_PLAYBACK_PWM in an application. This example shows how this driver can be used for playing a 16 bit uncompressed PCM audio from a single input buffer.

```
int16_t play_buffer[AUDIO_EXAMPLE_LENGTH];
uint32_t g_audio_callback_counter = 0;
void g_audio_example_counter_callback (audio_playback_callback_args_t * p_args)
{
 if (AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE == (p_args->event))
    {
       g_audio_callback_counter++;
    }
}
void basic_example (void)
{
```

```
 fsp_err_t err;
 /* Initialize the Audio Playback module for playing an audio stream. */
    err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the 16 Bit PCM audio stream to play next */
    err = RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, play_buffer,
AUDIO_EXAMPLE_LENGTH);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start to the play the selected audio stream*/
    err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait till the playback is completed */
 while (g_audio_callback_counter == 0)
    {
       ;
    }
 /* Stop playing. */
    err = RM_AUDIO_PLAYBACK_PWM_Stop(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### Streaming Example

This is an example of using Audio Playback module to play audio stream. This application uses a double buffer to store PCM sine wave data. It starts playing in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart playing in the main loop. This example also demonstrates conversion of signed PCM format data to unsigned PWM format data along with scaling the data samples for optimum PWM wave generation.

```
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_PERIOD_VALUE_AT_22050HZ (0x11B7U)
```

```
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800U)
#define AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER (1024U)
#define AUDIO_PLAYBACK_PWM_EXAMPLE_CONVERT_TO_PWM_SAMPLES (32768U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_DATA_BIT_SIZE (16U)
int16_t       g_stream_src[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
q15_t         g_pwm_sample[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
q15_t         g_pwm_scaled_sample[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when Audio Playback is ready for more data. */
void rm_audio_playback_example_callback (audio_playback_callback_args_t * p_args)
{
 /* Start playing next stream if data is ready. */
 if (AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE == (p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload data and handle errors. */
            rm_audio_playback_example_play();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
      }
    }
}
/* Load the next stream and check for error condition. */
void rm_audio_playback_example_play (void)
{
 /* Set the playback stream */
 fsp_err_t err;
```

```
    err =
 RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, (int16_t *)
&g_pwm_scaled_sample[g_buffer_index][0],

(AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof(int16_t)));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
       g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer. */
       g_data_ready = false;
    }
 else
    {
 /* The
  * application must wait until the audio playback is completed. In this example, the
  * callback sets data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void rm_audio_playback_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
 /* Create a sine wave. Using formula sample = sin(2 * pi * tone_frequency * t /
sampling_frequency) */
    uint32_t freq = AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK; i
+= 1)
    {
 float input = (((float) (freq * t)) * (float) M_TWOPI) /
```

```
AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       t++;
 /* Store sample. */
       int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
       g_stream_src[buffer_index][i] = sample;
    }
 /* Convert signed PCM data to unsigned PCM data as PWM needs unsigned input. */
    arm_offset_q15(&g_stream_src[buffer_index][0],
                   (q15_t) (INT16_MAX + 1),
                   &g_pwm_sample[buffer_index][0],
                   AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK);
 /* Scale the data by the selected period for the timer (calculated for equivalent
playback rate) */
    arm_scale_q15(&g_pwm_sample[buffer_index][0],
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_PERIOD_VALUE_AT_22050HZ,
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_DATA_BIT_SIZE,
                  &g_pwm_scaled_sample[buffer_index][0],
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK);
 /* Data is ready to be sent in the interrupt. */
    g_data_ready = true;
}
void rm_audio_playback_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module.
  * Configure the following pins in the pin configurator for PWM output:
  * - If the GPT timer is used for generation of PWM waves configure GTIOCAn or
GTIOCBn pin and enable the output
  * to these pins through the GPT module properties for desired channel n.
  * - Otherwise, if AGT is used for generation of PWM waves configure AGTOAn or
AGTOBn pin and enable the output to
  * to these pins through the AGT module properties for desired channel n.
  * Configure the DMAC/DTC destination address as following:
  * - If the GPT timer is used for generation of PWM waves, configure DMAC/DTC
```

```
destination address to the address of
  * GTCCRC register (&R_GPTn->GTCCR[2]) if PWM output pin is GTIOCA otherwise
configure to the address of GTCCRD
  * register (&R_GPTn->GTCCR[3]) if PWM output pin is GTIOCB for desired GPT channel
n.
  * - If the AGT timer is used for generation of PWM waves, configure DMAC/DTC
destination address as the address of
  * AGTCMA register (&R_AGTn->AGTCMA) if PWM output pin is AGTOA otherwise the
address of AGTCMB register
  * (&R_AGTn->AGTCMB) if the PWM output pin is AGTOB for desired AGT channel n.
  * Configure the DMAC/DTC transfer size as 4 Bytes if PWM interface is GPT timer
otherwise configure transfer size as 2 Bytes if
  * PWM interface is AGT timer. */
    err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start timer and transfer modules. */
    err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       rm_audio_playback_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
     {
 /* Clear flag. */
           g_send_data_in_main_loop = false;
 /* Reload data and handle errors. */
           rm_audio_playback_example_play();
     }
```

```
/* If the next buffer is ready, wait for the data to be sent in the interrupt. */
while (g_data_ready)
     {
/* Do nothing. */
     }
   }
}
```

### Data Structures

| | |
|---|---|
| struct | audio_playback_pwm_instance_ctrl_t |

## Data Structure Documentation

### ◆ audio_playback_pwm_instance_ctrl_t

| struct audio_playback_pwm_instance_ctrl_t | |
|---|---|
| AUDIO_PLAYBACK_PWM instance control block. DO NOT MODIFY. Initialization occurs when RM_AUDIO_PLAYBACK_PWM_Open() is called. | |
| **Data Fields** | |
| void(* | p_callback )(audio_playback_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| audio_playback_cfg_t const * | p_cfg |
| | Pointer to the configuration structure. |
| | |
| uint32_t | open |
| | Used by driver to check if the control structure is valid. |
| | |
| timer_instance_t const * | p_lower_lvl_timer |
| | Timer API used to generate sampling frequency and GPT/AGT API used to access PWM hardware. |
| | |
| transfer_instance_t const * | p_lower_lvl_transfer |
| | Transfer API used to transfer data each sampling frequency. |

## Field Documentation

### ◆ p_callback

void(* audio_playback_pwm_instance_ctrl_t::p_callback) (audio_playback_callback_args_t *p_args)

Callback called when play is complete.

### ◆ p_context

void* audio_playback_pwm_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in audio_playback_callback_args_t.

## Function Documentation

### ◆ RM_AUDIO_PLAYBACK_PWM_Open()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Open ( audio_playback_ctrl_t *const *p_api_ctrl*, audio_playback_cfg_t const *const *p_cfg* )

Opens and configures the Audio Playback with PWM driver. Sets playback speed and transfer rate to read the audio buffer.

Example:

```
/* Initialize the Audio Playback module for playing an audio stream. */
   err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Audio Playback module successfully configured. |
| FSP_ERR_ALREADY_OPEN | Module already open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL or callback is NULL. |

### ◆ RM_AUDIO_PLAYBACK_PWM_Close()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Close ( audio_playback_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | Pointer pointing to NULL. |

***Note***

*This function will close all the lower level HAL drivers as well.*

### ◆ RM_AUDIO_PLAYBACK_PWM_Start()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Start ( audio_playback_ctrl_t *const *p_api_ctrl*)

Start the PWM HAL driver (AGT or GPT) and timer HAL (AGT or GPT) drivers.

- Example:

```
/* Start to the play the selected audio stream*/
   err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Audio playback hardware started successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver not open. This function calls<br>• timer_api_t::start |

### ◆ RM_AUDIO_PLAYBACK_PWM_Stop()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Stop ( audio_playback_ctrl_t *const *p_api_ctrl*)

Stop the PWM HAL driver (AGT or GPT) and timer HAL driver (AGT or GPT).

- Example:

```
/* Stop playing. */
   err = RM_AUDIO_PLAYBACK_PWM_Stop(&g_audio_playback_pwm_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Audio playback hardware stopped successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver not open. This function calls<br>• timer_api_t::stop |

### ◆ RM_AUDIO_PLAYBACK_PWM_Play()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Play ( audio_playback_ctrl_t *const *p_api_ctrl*, void const *const *p_buffer*, uint32_t *length* )

Play a single audio buffer by input samples to the PWM HAL (AGT or GPT) at the sampling frequency configured by the timer.

- Example:

```
/* Set the 16 Bit PCM audio stream to play next */
   err = RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, play_buffer,
AUDIO_EXAMPLE_LENGTH);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Buffer playback began successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl or p_buffer is NULL or buffer length is greater than 0x10000. |
| FSP_ERR_NOT_OPEN | Driver not open.. This function calls<br>• transfer_api_t::reset |

#### 5.2.3.3 Minimp3 Library

Modules » Audio

Minimalistic, single-header library for decoding MP3.

# Overview

Integration of minimp3 library with FSP RA.

# Usage

For detailed usage of minimp3, please visit https://github.com/lieff/minimp3.

**ARM Helium Support**

Helium (with SIMD support) can be enabled to accelerate the decoding process on devices that

support it.

**Limitations**

**Minimp3** requires a MCU with 32KB of RAM size or more to function properly.

Minimum stack usage is 18KB that must be considered the stack size under BSP configuration tab to ensure provide enough stack for the software operation.

This library uses floating-point numbers in its operation, therefore it shall be used on MCUs that have FPU. Non-FPUs are still able to use this library but the performance will be significantly degraded.

**How to use**

To use minimp3 library, follow the steps below:

- In e2studio, add minimp3 stack by selecting **New Stack -> Audio -> Minimp3 Library Source**.
- Fix stack constraints and choose **Generate Project Content**.
- In your source code, remember to include **minimp3.h** or **minimp3_ex.h**.

# Configuration

### Build Time Configurations for Lib

The following build time configurations are defined in fsp_cfg/minimp3/minimp3_config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Float Output | • Enabled<br>• Disabled | Disabled | Output PCM as float instead of short |
| SIMD Optimization | • Enabled<br>• Disabled | Disabled | Enable SIMD optimization for MP3 decoding using Arm Helium (MVE). |
| Decode MP3 only | • Enabled<br>• Disabled | Disabled | Enable this option to skip MP1/MP2 and only support MP3 decoding. |

# Examples

### Minimp3 Basic Example

This is a basic example demonstrating minimal usage of the minimp3 library implementation in an application.

```
void minimp3_basic_example ()
{
    uint32_t            mp3_offset = 0;
    uint32_t            pcm_offset = 0;
```

```
    mp3dec_frame_info_t frame_info;
    mp3dec_t            mp3dec;
 /* Initialize for decoding a new position */
    mp3dec_init(&mp3dec);
 /* Decode MP3 data frame by frame */
 do
    {
 /* Decode a frame */
 int samples = mp3dec_decode_frame(&mp3dec,
                                        g_mp3_buf + mp3_offset,
                                        (int) (MP3_BUFFER_SIZE_BYTES - mp3_offset),
                                        g_pcm_buf + pcm_offset,
                                        &frame_info);
 /* Update position offset of MP3 and PCM data */
       mp3_offset += (uint32_t) (frame_info.frame_bytes);
       pcm_offset += (uint32_t) (samples * frame_info.channels);
    } while (MP3_BUFFER_SIZE_BYTES != mp3_offset);
}
```

### Minimp3 Streaming Example

This example demonstrates streaming usage of the minimp3 library implementation in an application.

```
void minimp3_streaming_example ()
{
    uint8_t          * p_mp3_feed_pos;
    uint32_t            mp3_feed_size;
    mp3d_sample_t       pcm_data[MINIMP3_MAX_SAMPLES_PER_FRAME];
    mp3dec_frame_info_t frame_info;
    mp3dec_t            mp3dec;
 /* Initialize for decoding a new position */
    mp3dec_init(&mp3dec);
 /* Stream MP3 data frame by frame */
 do
```

```
    {
 /* Update MP3 data position and MP3 data size.
  * This is a user-defined function for providing continuous MP3 data for streaming.
*/
       mp3_read(&p_mp3_feed_pos, &mp3_feed_size);
 /* Decode one frame */
 int samples = mp3dec_decode_frame(&mp3dec, p_mp3_feed_pos, (int) mp3_feed_size,
pcm_data, &frame_info);
 /* Get PCM data output size. */
       uint32_t pcm_size = (uint32_t) (samples * frame_info.channels);
 /* Handle PCM data output.
  * This is a user-defined function for handling PCM data output after each frame is
decoded. */
       pcm_progress(pcm_data, pcm_size);
    } while (mp3_feed_size);
}
```

### 5.2.3.4 Pulse Density Modulation Interface (r_pdm)

Modules » Audio

**Functions**

| | |
|---|---|
| fsp_err_t | R_PDM_Open (pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg) |
| fsp_err_t | R_PDM_Start (pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback) |
| fsp_err_t | R_PDM_Stop (pdm_ctrl_t *const p_ctrl) |
| fsp_err_t | R_PDM_SoundDetectionEnable (pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t sound_detection_setting) |
| fsp_err_t | R_PDM_SoundDetectionDisable (pdm_ctrl_t *const p_ctrl) |
| fsp_err_t | R_PDM_Read (pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_PDM_StatusGet (pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status) |

| | |
|---|---|
| fsp_err_t | R_PDM_Close (pdm_ctrl_t *const p_ctrl) |
| fsp_err_t | R_PDM_CallbackSet (pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context, pdm_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the PDM peripheral on RA MCUs. This module implements the PDM Interface.

# Overview

The Pulse Density Modulation Interface (PDM-IF) has channels that are connectable with external microphone which outputs the pulse density modulated signal. PDM-IF can filter and convert 1-bit digital data streams that were pulse density modulated at a high sampling rate into 20-bit or 16-bit digital data at a lower sampling rate.

### Features

- Capable of filtering 1-bit digital input data and converting them into 20-bit or 16-bit digital data. The bit order is little-endian.
- Sound detection can be enabled for each channel. A moving average filter can be enabled for sound detection. Sound detection can be used to wake up the MCU from software standby mode.
- Each channel of IP includes programmable filters: 4th order sinc filter, high-pass filter (for suppression of DC bias), correction filter (for sinc passband distortion), and half-band decimation filter (for aliasing distortion). – IP supports programmable and flexible decimation ratios. – The sinc filter is selectable as first-, second-, third-, fourth-order.
- IP supports DMA operation through APB. – Each channel has a 32 stage FIFO for receiving converted PDM data.
- Configurable callback for each channel which notifies of received data based on a configurable threshold as well as sound detection and error events.

# Configuration

### Build Time Configurations for r_pdm

The following build time configurations are defined in fsp_cfg/r_pdm_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DMAC Support | • Enabled<br>• Disabled | Disabled | If enabled code for DMAC transfer support is included in the build. |

### Configurations for Audio > PDM (r_pdm)

This module can be added to the Stacks tab via New Stack > Audio > PDM (r_pdm).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Input | | | |
| Edge | • Rising<br>• Falling | Rising | Select the edge of input audio data. |
| PDM_CLK Frequency | Refer to the RA Configuration tool for available options. | module.driver.pdm.clock_div.clock_div_2 | Select PDM_CLK Frequency (Hz). |
| Output | | | |
| Bit Format | • 20 Bits mode {1 Sign bit and [18:0] bits of the buffer}<br>• 20 Bits mode {2 Sign bits and [18:1] bits of the buffer}<br>• 20 Bits mode {3 Sign bits and [18:2] bits of the buffer}<br>• 20 Bits mode {4 Sign bits and [18:3] bits of the buffer}<br>• 16 Bits mode {1 Sign bit and [18:4] bits of the buffer}<br>• 16 Bits mode {1 Sign bit and [17:3] bits of the buffer}<br>• 16 Bits mode {1 Sign bit and [16:2] bits of the buffer}<br>• 16 Bits mode {1 Sign bit and [15:1] bits of the buffer}<br>• 16 Bits mode {1 Sign bit and [14:0] bits of the buffer} | module.driver.pdm.pcm_width.pcm_width_20bits_0_18 | Select the bit format of the PCM data stored into the data FIFO. |
| Target Sampling Frequency (Hz) | Manual Entry | 32000 | Select Target sampling Frequency (Hz). |
| Filter | | | |

Filter > Low-pass Filter

| | | | |
|---|---|---|---|
| Low-pass Filter Input Shift | • No shift<br>• 1-bit right shift<br>• 2-bits right shift<br>• 3-bits right shift | No shift | Select the low-pass (half-band decimation) filter input shift setting. |
| LPF Coefficient h0(10) | Value must be an integer between 0x0000 and 0x1FFF | 0x0400 | Specify the Low-Pass Filter Coefficient h0(10) |
| LPF Coefficient h1(0) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FF8 | Specify the Low-Pass Filter Coefficient h1(0) |
| LPF Coefficient h1(1) | Value must be an integer between 0x0000 and 0x1FFF | 0x000A | Specify the Low-Pass Filter Coefficient h1(1) |
| LPF Coefficient h1(2) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FF0 | Specify the Low-Pass Filter Coefficient h1(2) |
| LPF Coefficient h1(3) | Value must be an integer between 0x0000 and 0x1FFF | 0x0018 | Specify the Low-Pass Filter Coefficient h1(3) |
| LPF Coefficient h1(4) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FDC | Specify the Low-Pass Filter Coefficient h1(4) |
| LPF Coefficient h1(5) | Value must be an integer between 0x0000 and 0x1FFF | 0x0034 | Specify the Low-Pass Filter Coefficient h1(5) |
| LPF Coefficient h1(6) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FB3 | Specify the Low-Pass Filter Coefficient h1(6) |
| LPF Coefficient h1(7) | Value must be an integer between 0x0000 and 0x1FFF | 0x0076 | Specify the Low-Pass Filter Coefficient h1(7) |
| LPF Coefficient h1(8) | Value must be an integer between 0x0000 and 0x1FFF | 0x1F2E | Specify the Low-Pass Filter Coefficient h1(8) |
| LPF Coefficient h1(9) | Value must be an integer between 0x0000 and 0x1FFF | 0x0289 | Specify the Low-Pass Filter Coefficient h1(9) |
| LPF Coefficient h1(10) | Value must be an integer between 0x0000 and 0x1FFF | 0x0289 | Specify the Low-Pass Filter Coefficient h1(10) |
| LPF Coefficient h1(11) | Value must be an integer between 0x0000 and 0x1FFF | 0x1F2E | Specify the Low-Pass Filter Coefficient h1(11) |
| LPF Coefficient h1(12) | Value must be an | 0x0076 | Specify the Low-Pass |

| | | | |
|---|---|---|---|
| | integer between 0x0000 and 0x1FFF | | Filter Coefficient h1(12) |
| LPF Coefficient h1(13) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FB3 | Specify the Low-Pass Filter Coefficient h1(13) |
| LPF Coefficient h1(14) | Value must be an integer between 0x0000 and 0x1FFF | 0x0034 | Specify the Low-Pass Filter Coefficient h1(14) |
| LPF Coefficient h1(15) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FDC | Specify the Low-Pass Filter Coefficient h1(15) |
| LPF Coefficient h1(16) | Value must be an integer between 0x0000 and 0x1FFF | 0x0018 | Specify the Low-Pass Filter Coefficient h1(16) |
| LPF Coefficient h1(17) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FF0 | Specify the Low-Pass Filter Coefficient h1(17) |
| LPF Coefficient h1(18) | Value must be an integer between 0x0000 and 0x1FFF | 0x000A | Specify the Low-Pass Filter Coefficient h1(18) |
| LPF Coefficient h1(19) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FF8 | Specify the Low-Pass Filter Coefficient h1(19) |
| Filter > Compensation Filter | | | |
| Compensation Filter Input Shift | • No shift<br>• 1-bit right shift<br>• 2-bits right shift<br>• 3-bits right shift | No shift | Select the compensation filter input shift setting. |
| Compensation Filter Coefficient h(0) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FE8 | Specify the Compensation Filter Coefficient h(0) |
| Compensation Filter Coefficient h(1) | Value must be an integer between 0x0000 and 0x1FFF | 0x0039 | Specify the Compensation Filter Coefficient h(1) |
| Compensation Filter Coefficient h(2) | Value must be an integer between 0x0000 and 0x1FFF | 0x003C | Specify the Compensation Filter Coefficient h(2) |
| Compensation Filter Coefficient h(3) | Value must be an integer between 0x0000 and 0x1FFF | 0x1E56 | Specify the Compensation Filter Coefficient h(3) |
| Compensation Filter Coefficient h(4) | Value must be an integer between 0x0000 and 0x1FFF | 0x01DC | Specify the Compensation Filter Coefficient h(4) |
| Compensation Filter Coefficient h(5) | Value must be an integer between | 0x06E1 | Specify the Compensation Filter |

| | | | |
|---|---|---|---|
| | 0x0000 and 0x1FFF | | Coefficient h(5) |
| Compensation Filter Coefficient h(6) | Value must be an integer between 0x0000 and 0x1FFF | 0x01DC | Specify the Compensation Filter Coefficient h(6) |
| Compensation Filter Coefficient h(7) | Value must be an integer between 0x0000 and 0x1FFF | 0x1E56 | Specify the Compensation Filter Coefficient h(7) |
| Compensation Filter Coefficient h(8) | Value must be an integer between 0x0000 and 0x1FFF | 0x003C | Specify the Compensation Filter Coefficient h(8) |
| Compensation Filter Coefficient h(9) | Value must be an integer between 0x0000 and 0x1FFF | 0x0039 | Specify the Compensation Filter Coefficient h(9) |
| Compensation Filter Coefficient h(10) | Value must be an integer between 0x0000 and 0x1FFF | 0x1FE8 | Specify the Compensation Filter Coefficient h(10) |
| Filter > High-pass Filter | | | |
| High-pass Filter Input Shift | • No shift<br>• 1-bit right shift<br>• 2-bits right shift<br>• 3-bits right shift | No shift | Select the high-pass filter input shift setting. |
| HPF Coefficient s(0) | Value must be an integer between 0x0000 and 0xFFFF | 0x3F61 | Specify the High-Pass Filter Coefficient s(0) |
| HPF Coefficient k(1) | Value must be an integer between 0x0000 and 0xFFFF | 0x3EC1 | Specify the High-Pass Filter Coefficient k(1) |
| HPF Coefficient h(0) | Value must be an integer between 0x0000 and 0xFFFF | 0x4000 | Specify the High-Pass Filter Coefficient h(0) |
| HPF Coefficient h(1) | Value must be an integer between 0x0000 and 0xFFFF | 0xC000 | Specify the High-Pass Filter Coefficient h(1) |
| Moving Average Mode | • 1-order (filter is skipped)<br>• 2-order<br>• 4-order | 1-order (filter is skipped) | Select the moving average mode for sound detection data. |
| Sinc Filter Mode | • 1-order<br>• 2-order<br>• 3-order<br>• 4-order | 4-order | Select the sinc filter mode setting. |
| Data Reception | | | |
| Buffer Overwrite Detection Enable | • Enabled<br>• Disabled | Enabled | Select buffer overwrite detection. |
| Data Reception | • 1 or more data | 16 or more data | Select Data reception |

| | | | |
|---|---|---|---|
| Interrupt Threshold | • 2 or more data<br>• 4 or more data<br>• 8 or more data<br>• 16 or more data | | interrupt threshold. |
| Short Circuit Detection | | | |
| Short Circuit Detection Enable | • Enabled<br>• Disabled | Disabled | Select short circuit detection. |
| Short Circuit Detection High Continuous Detection Count | Value must be an integer between 0x0000 and 0x1FFF | 0x0 | Specify the Short circuit detection High Continuous detection count. |
| Short Circuit Detection Low Continuous Detection Count | Value must be an integer between 0x0000 and 0x1FFF | 0x0 | Short circuit detection Low Continuous detection count. |
| Overvoltage Detection | | | |
| Overvoltage Detection Lower Limit Enable | • Enabled<br>• Disabled | Disabled | Select overvoltage detection lower limit. |
| Overvoltage Detection Lower Limit | Value must be an integer between 0x0000 and 0xFFFFF | 0x0 | Specify the Overvoltage detection lower limit |
| Overvoltage Detection Upper Limit Enable | • Enabled<br>• Disabled | Disabled | Select overvoltage detection upper limit. |
| Overvoltage Detection Upper Limit | Value must be an integer between 0x0000 and 0xFFFFF | 0x0 | Specify the Overvoltage detection upper limit |
| Interrupt | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR). |
| Sound Detection Interrupt Priority | MCU Specific Options | | Select the sound detection interrupt priority. |
| Data Reception Interrupt Priority | MCU Specific Options | | Select the data reception interrupt priority. |
| Error Detection Interrupt Priority | MCU Specific Options | | Select the error detection interrupt priority. |
| Name | Name must be a valid C symbol | g_pdm0 | Module name. |

| | | | |
|---|---|---|---|
| Channel | Value must be an non-negative integer | 0 | Specify the PDM channel. |

# Usage Notes

### Startup and Settling Time

Upon calling R_PDM_Open the user must wait for the filters to settle and any connected microphones to startup, otherwise PDM data may not be correct upon calling R_PDM_Start.

A macro G_PDMn_FILTER_SETTLING_TIME_US (where n is the channel number) is generated for the settling time. The user can use this macro to delay for the settling time via their preferred method of doing a delay.

In addition to the filter settling time any connected microphone startup time should be accounted for. This can be found in the connected microphone's datasheet.

### Audio Data

PDM hardware converts PDM data to PCM data.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

| Bit Depth | Array Data Type | Required Alignment |
|---|---|---|
| 16 Bits | 32-bit integer, right justified | 4 byte alignment |
| 20 Bits | 32-bit integer, right justified | 4 byte alignment |

### Data Format for 16 Bits

| Bit | Description |
|---|---|
| 31-20 | Fixed to 0 |
| 19-16 | Sign bits |
| 15-0 | Fixed-point data |

### Data Format for 20 Bits

| Bit | Description |
|---|---|
| 31-20 | Fixed to 0 |
| 19-0 | Fixed-point data |

### Modes of Operation

In addition to calling R_PDM_Open, other APIs need to be called to activate certain features of PDM. The following table lists what each API controls:

| API | Features Enabled by API |
|---|---|
| R_PDM_Open | over/undervoltage detection, short circuit detection |

| | |
|---|---|
| R_PDM_Start | data reception via interrupts or DMAC depending configuration, error interrupts |
| R_PDM_SoundDetectionEnable | sound detection, sound detection thresholds, error interrupts |

### Error Interrupts

Error interrupts are only enabled when data reception is enabled (R_PDM_Start has been called) OR sound detection is enabled (R_PDM_SoundDetectionEnable has been called).

R_PDM_StatusGet can be called to get error statuses if error interrupts are not enabled (e.g. if only R_PDM_Open has been called).

### Low Power Mode

*Note*

*In order to use PDM during low power mode MOCO must be enabled for software standby mode (see MOCOSCR register).*

Some microphones support going into a low power mode by changing the PDM clock to a low frequency. In order to do this with r_pdm the module instance should be closed and reopened with a separate configuration that uses a lower PDM clock.

### Limitations

- Stereo mode is not fully supported. Two r_pdm driver instances can be used in combination with the pcm_edge configuration to get left/right stereo output from an appropriately configured PDM data stream, however, the data from these two instances will not be synchronized. Full synchronization of PDM channels requires using PDM mirror registers and starting data reading at the same time between channels. In the future a stereo-specific driver may be implemented using the r_pdm driver as a lower level driver to cover this use case.
- When calling R_PDM_Start any data in the FIFO before DATRE is enabled will be discarded.

## Examples

### Basic Example

```
#define PDM_BUFFER_NUM_SAMPLES 1024 // Number of samples to store in data buffer
#define PDM_CALLBACK_NUM_SAMPLES PDM_BUFFER_NUM_SAMPLES / 2 // Number of samples to
store in data buffer before getting a callback
#define PDM_MIC_STARTUP_TIME_US 35000 // PDM microphone startup time, RA8P1_EK mic
startup time is 35ms
#define PDM_SDE_UPPER_LIMIT 5000 // Detect sound above this value
#define PDM_SDE_LOWER_LIMIT 0xFFF80000 // Detect sound below this value
uint32_t g_pdm0_buffer[PDM_BUFFER_NUM_SAMPLES];
void r_pdm_basic_messaging_core0_example (void)
{
```

```
 /* Open PDM instance. */
 fsp_err_t err = R_PDM_Open(&g_pdm0_ctrl, &g_pdm0_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for PDM filters to settle. Use settling time calculated by e2 studio.
  * Also wait for PDM mic to startup. */
 R_BSP_SoftwareDelay(G_PDM0_FILTER_SETTLING_TIME_US + PDM_MIC_STARTUP_TIME_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable sound detection (if desired). */
 pdm_sound_detection_setting_t sound_detection_setting =
    {
       .sound_detection_lower_limit = PDM_SDE_LOWER_LIMIT,
       .sound_detection_upper_limit = PDM_SDE_UPPER_LIMIT
    };
 fsp_err_t err = R_PDM_SoundDetectionEnable(&g_pdm0_ctrl, sound_detection_setting);
    assert(FSP_SUCCESS == err);
 /* Start receiving PDM data. */
    err = R_PDM_Start(&g_pdm0_ctrl, g_pdm0_buffer, sizeof(g_pdm0_buffer),
PDM_CALLBACK_NUM_SAMPLES);
    assert(FSP_SUCCESS == err);
 /* DATA PROCESSING HERE */
 /* Stop receiving PDM data. */
    err = R_PDM_Stop(&g_pdm0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Close PDM instance. */
    err = R_PDM_Close(&g_pdm0_ctrl);
    assert(FSP_SUCCESS == err);
}
void pdm_example_callback (pdm_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case PDM_EVENT_SOUND_DETECTION:
      {
 /* Sound has been detected based on configured thresholds. */
```

```
break;
      }
case PDM_EVENT_DATA:
     {
/* PDM_CALLBACK_NUM_SAMPLES has been stored in g_pdm0_buffer. */
break;
      }
case PDM_EVENT_ERROR:
     {
if (p_args->error & PDM_ERROR_SHORT_CIRCUIT)
     {
/* Short circuit has been detected based on configured counts */
        }
if (p_args->error & PDM_ERROR_OVERVOLTAGE_LOWER)
     {
/* Overvoltage has been detected based on configured lower limit */
        }
if (p_args->error & PDM_ERROR_OVERVOLTAGE_UPPER)
     {
/* Overvoltage has been detected based on configured upper limit */
        }
if (p_args->error & PDM_ERROR_BUFFER_OVERWRITE)
     {
/* Buffer overwrite has occurred */
        }
break;
      }
default:
     {
break;
      }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | pdm_instance_ctrl_t |
| struct | pdm_extended_cfg_t |

### Macros

| | |
|---|---|
| #define | PDM_NUM_HPF_COEFFICIENT_H<br>Number of high pass filter h coefficients. |
| #define | PDM_NUM_COMPENSATION_FILTER_COEFFICIENT_H<br>Number of compensation filter h coefficients. |
| #define | PDM_NUM_LPF_FILTER_COEFFICIENT_H1<br>Number of low pass filter h1 coefficients. |

### Enumerations

| | |
|---|---|
| enum | pdm_clk_div_t |
| enum | pdm_moving_average_mode_t |
| enum | pdm_low_pass_filter_shift_t |
| enum | pdm_compensation_filter_shift_t |
| enum | pdm_high_pass_filter_shift_t |
| enum | pdm_sinc_filter_mode_t |
| enum | pdm_interrupt_threshold_t |
| enum | pdm_short_circuit_enable_t |
| enum | pdm_overvoltage_lower_limit_enable_t |
| enum | pdm_overvoltage_upper_limit_enable_t |
| enum | pdm_buffer_overwrite_detection_enable_t |

## Data Structure Documentation

### ◆ pdm_instance_ctrl_t

| struct pdm_instance_ctrl_t |
|---|
| Channel instance control block. DO NOT INITIALIZE. Initialization occurs when pdm_api_t::open is called. |

### ◆ pdm_extended_cfg_t

| struct pdm_extended_cfg_t | | |
|---|---|---|
| PDM configuration extension. This extension is optional. | | |
| Data Fields | | |
| pdm_clk_div_t | clock_div | PDM_CLKn's division ratio to core clock. |
| pdm_short_circuit_enable_t | short_circuit_detection_enable | Short circuit detection enable.<br><br>Function Settings. |
| pdm_overvoltage_lower_limit_enable_t | over_voltage_lower_limit_detection_enable | Overvoltage detection lower limit enable. |
| pdm_overvoltage_upper_limit_enable_t | over_voltage_upper_limit_detection_enable | Overvoltage detection upper limit enable. |
| pdm_buffer_overwrite_detection_enable_t | buffer_overwrite_detection_enable | Buffer overwrite detection enable. |
| pdm_moving_average_mode_t | moving_average_mode | Moving average mode f or sound detection data.<br><br>Filter Settings. |
| pdm_low_pass_filter_shift_t | low_pass_filter_shift | Low-pass (half -band decimation) filter input shift setting. |
| pdm_compensation_filter_shift_t | compensation_filter_shift | Compensation filter input shift setting. |
| pdm_high_pass_filter_shift_t | high_pass_filter_shift | High-pass filter input shift setting. |
| pdm_sinc_filter_mode_t | sinc_filter_mode | Sinc filter mode setting. |
| uint8_t | sincrng | Sinc filter output valid range. |
| uint8_t | sincdec | Sinc filter decimation ratio. |
| uint16_t | hpf_coefficient_s0 | High-pass filter coefficient s0 (16-bit signed fixed point data) |
| uint16_t | hpf_coefficient_k1 | High-pass filter coefficient k1 (16-bit signed fixed point data) |
| uint16_t | hpf_coefficient_h[ PDM_NUM_HPF_COEFFICIENT_H] | High-pass filter coefficient h(p) (16-bit signed fixed point data) |
| uint16_t | compensation_filter_coefficient_h[ PDM_NUM_COMPENSATION_FILTER_COEFFICIENT_H] | High-pass filter coefficient h(p) (13-bit signed fixed point data) |
| uint16_t | lpf_coefficient_h0 | Low-pass (half-band |

| | | |
|---|---|---|
| | | decimation) filter coefficient h0 (13-bit signed fixed point data) |
| uint16_t | lpf_coefficient_h1[PDM_NUM_LPF_FILTER_COEFFICIENT_H1] | Low-pass (half-band decimation) filter coefficient h1 (13-bit signed fixed point data) |
| pdm_interrupt_threshold_t | interrupt_threshold | Data reception interrupt threshold.<br><br>Data Reception. |
| uint16_t | short_circuit_count_h | Short circuit detection High Continuous detection count.<br><br>Short-Circuit Detection. |
| uint16_t | short_circuit_count_l | Continuous detection count. |
| uint32_t | overvoltage_detection_lower_limit | Overvoltage detection lower limit (20-bit signed fixed point data)<br><br>Overvoltage Detection. |
| uint32_t | overvoltage_detection_upper_limit | Overvoltage detection upper limit (20-bit signed fixed point data) |

### Enumeration Type Documentation

### ◆ pdm_clk_div_t

| enum pdm_clk_div_t | |
|---|---|
| PDM_CLKn's division ratio to core clock. | |
| Enumerator | |
| PDM_CLOCK_DIV_2 | 0000b: 1/2 |
| PDM_CLOCK_DIV_4 | 0001b: 1/4 |
| PDM_CLOCK_DIV_6 | 0010b: 1/6 |
| PDM_CLOCK_DIV_8 | 0011b: 1/8 |
| PDM_CLOCK_DIV_10 | 0100b: 1/10 |
| PDM_CLOCK_DIV_12 | 0101b: 1/12 |
| PDM_CLOCK_DIV_14 | 0110b: 1/14 |
| PDM_CLOCK_DIV_16 | 0111b: 1/16 |
| PDM_CLOCK_DIV_18 | 1000b: 1/18 |
| PDM_CLOCK_DIV_20 | 1001b: 1/20 |
| PDM_CLOCK_DIV_22 | 1010b: 1/22 |
| PDM_CLOCK_DIV_24 | 1011b: 1/24 |
| PDM_CLOCK_DIV_26 | 1100b: 1/26 |
| PDM_CLOCK_DIV_28 | 1101b: 1/28 |
| PDM_CLOCK_DIV_30 | 1110b: 1/30 |
| PDM_CLOCK_DIV_32 | 1111b: 1/32 |

### ◆ pdm_moving_average_mode_t

| enum pdm_moving_average_mode_t | |
|---|---|
| Moving average mode for sound detection data. | |
| Enumerator | |
| PDM_MOVING_AVERAGE_MODE_1_ORDER | 1-order |
| PDM_MOVING_AVERAGE_MODE_2_ORDER | 2-order |
| PDM_MOVING_AVERAGE_MODE_4_ORDER | 4-order |

### ◆ pdm_low_pass_filter_shift_t

| enum pdm_low_pass_filter_shift_t | |
|---|---|
| Low-pass (half-band decimation) filter input shift setting. | |
| Enumerator | |
| PDM_LPF_RIGHT_SHIFT_0 | No shift. |
| PDM_LPF_RIGHT_SHIFT_1 | 1-bit right shift |
| PDM_LPF_RIGHT_SHIFT_2 | 2-bit right shift |
| PDM_LPF_RIGHT_SHIFT_3 | 3-bit right shift |

### ◆ pdm_compensation_filter_shift_t

| enum pdm_compensation_filter_shift_t | |
|---|---|
| Compensation filter input shift setting. | |
| Enumerator | |
| PDM_COMPENSATION_FILTER_RIGHT_SHIFT_0 | No shift. |
| PDM_COMPENSATION_FILTER_RIGHT_SHIFT_1 | 1-bit right shift |
| PDM_COMPENSATION_FILTER_RIGHT_SHIFT_2 | 2-bit right shift |
| PDM_COMPENSATION_FILTER_RIGHT_SHIFT_3 | 3-bit right shift |

### ◆ pdm_high_pass_filter_shift_t

| enum pdm_high_pass_filter_shift_t | |
|---|---|
| High-pass filter input shift setting. | |
| Enumerator | |
| PDM_HPF_RIGHT_SHIFT_0 | No shift. |
| PDM_HPF_RIGHT_SHIFT_1 | 1-bit right shift |
| PDM_HPF_RIGHT_SHIFT_2 | 2-bit right shift |
| PDM_HPF_RIGHT_SHIFT_3 | 3-bit right shift |

### ◆ pdm_sinc_filter_mode_t

| enum pdm_sinc_filter_mode_t | |
|---|---|
| Sinc filter mode setting. | |
| Enumerator | |
| PDM_SINC_FILTER_MODE_1 | 1-order |
| PDM_SINC_FILTER_MODE_2 | 2-order |
| PDM_SINC_FILTER_MODE_3 | 3-order |
| PDM_SINC_FILTER_MODE_4 | 4-order |

### ◆ pdm_interrupt_threshold_t

| enum pdm_interrupt_threshold_t | |
|---|---|
| Data reception interrupt threshold. | |
| Enumerator | |
| PDM_INTERRUPT_THRESHOLD_1 | 000b: Output interrupt when receiving 1 or more data |
| PDM_INTERRUPT_THRESHOLD_2 | 001b: Output interrupt when receiving 2 or more data |
| PDM_INTERRUPT_THRESHOLD_4 | 010b: Output interrupt when receiving 4 or more data |
| PDM_INTERRUPT_THRESHOLD_8 | 011b: Output interrupt when receiving 8 or more data |
| PDM_INTERRUPT_THRESHOLD_16 | 100b: Output interrupt when receiving 16 or more data |

### ◆ pdm_short_circuit_enable_t

| enum pdm_short_circuit_enable_t | |
|---|---|
| Short circuit detection enable setting. | |
| Enumerator | |
| PDM_SHORT_CIRCUIT_DISABLED | Short circuit disabled. |
| PDM_SHORT_CIRCUIT_ENABLED | Short circuit enabled. |

### ◆ pdm_overvoltage_lower_limit_enable_t

| enum pdm_overvoltage_lower_limit_enable_t | |
|---|---|
| Overvoltage lower limit enable setting. | |
| Enumerator | |
| PDM_OVERVOLTAGE_LOWER_LIMIT_DISABLED | Overvoltage lower limit disabled. |
| PDM_OVERVOLTAGE_LOWER_LIMIT_ENABLED | Overvoltage lower limit enabled. |

### ◆ pdm_overvoltage_upper_limit_enable_t

| enum pdm_overvoltage_upper_limit_enable_t | |
|---|---|
| Overvoltage upper limit enable setting. | |
| Enumerator | |
| PDM_OVERVOLTAGE_UPPER_LIMIT_DISABLED | Overvoltage upper limit disabled. |
| PDM_OVERVOLTAGE_UPPER_LIMIT_ENABLED | Overvoltage upper limit enabled. |

### ◆ pdm_buffer_overwrite_detection_enable_t

| enum pdm_buffer_overwrite_detection_enable_t | |
|---|---|
| Buffer overwrite detection enable setting. | |
| Enumerator | |
| PDM_BUFFER_OVERWRITE_DETECTION_DISABLED | Buffer overwrite detection disabled. |
| PDM_BUFFER_OVERWRITE_DETECTION_ENABLED | Buffer overwrite detection enabled. |

## Function Documentation

### ◆ R_PDM_Open()

fsp_err_t R_PDM_Open ( pdm_ctrl_t *const *p_ctrl*, pdm_cfg_t const *const *p_cfg* )

Opens the PDM. Implements pdm_api_t::open.

This function sets this clock divisor and the configurations specified in pdm_cfg_t.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Ready for PDM communication. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_cfg is null. |
| FSP_ERR_ALREADY_OPEN | The control block has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel number is not available on this device. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_PDM_Start()

fsp_err_t R_PDM_Start ( pdm_ctrl_t *const *p_ctrl*, void *const *p_buffer*, size_t const *buffer_size*, uint32_t const *number_of_data_to_callback* )

Starts PDM. Implements pdm_api_t::start.

This function enables reception, and enables any transfer instances used.

The PDM will start on the next frame boundary.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PDM communication stop request issued. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_INVALID_SIZE | The size of buffer is not a multiple of pdm_interrupt_threshold_t. |
| FSP_ERR_INVALID_ALIGNMENT | The alignment of argument buffer_size is not matched with hardware specification. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_PDM_Stop()

fsp_err_t R_PDM_Stop ( pdm_ctrl_t *const *p_ctrl*)

Stops PDM. Implements pdm_api_t::stop.

This function disables reception, and disables any transfer instances used.

The PDM will stop on the next frame boundary.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PDM communication stop request issued. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_PDM_SoundDetectionEnable()

fsp_err_t R_PDM_SoundDetectionEnable ( pdm_ctrl_t *const *p_ctrl*, pdm_sound_detection_setting_t *sound_detection_setting* )

Configures and enables sound detection. Implements pdm_api_t::soundDetectionEnable.

This function configures sound detection based on the passed in settings. It then enables sound detection.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Sound detection configured and enabled. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_PDM_SoundDetectionDisable()

fsp_err_t R_PDM_SoundDetectionDisable ( pdm_ctrl_t *const *p_ctrl*)

Disables sound detection. Implements pdm_api_t::soundDetectionDisable.

This function disables sound detection.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Sound detection disabled. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_PDM_Read()

fsp_err_t R_PDM_Read ( pdm_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes* )

This function is not supported. Implements pdm_api_t::read.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported. |

### ◆ R_PDM_StatusGet()

fsp_err_t R_PDM_StatusGet ( pdm_ctrl_t *const *p_ctrl*, pdm_status_t *const *p_status* )

Gets PDM status and stores it in provided pointer p_status. Implements pdm_api_t::statusGet.

Calling this function will clear any set error flags in the hardware.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored successfully. |
| FSP_ERR_ASSERTION | The p_ctrl or p_status parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_PDM_Close()

fsp_err_t R_PDM_Close ( pdm_ctrl_t *const *p_ctrl*)

Closes PDM. Implements pdm_api_t::close.

This function powers down the PDM and closes the lower level timer and transfer drivers if they are used.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_PDM_CallbackSet()

fsp_err_t R_PDM_CallbackSet ( pdm_ctrl_t *const *p_ctrl*, void(*)(pdm_callback_args_t *) *p_callback*, void *const *p_context*, pdm_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements pdm_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.4 Bootloader

Modules

### Detailed Description

Bootloader Modules.

### Modules

| | |
|---|---|
| MCUboot Port (rm_mcuboot_port) | MCUboot Port for RA MCUs. |

### 5.2.4.1 MCUboot Port (rm_mcuboot_port)

Modules » Bootloader

MCUboot Port for RA MCUs.

# Overview

*Note*

*The MCUboot Port does not provide any interfaces to the user. Consult the MCUboot documentation at https://mcu-tools.github.io/mcuboot/ for further information.*
*FSP 6.0.0 changes the way linker scripts are handled which in turn will change how the user configures MCUboot. Where applicable, the documentation below will have a separate section if the usage is markedly different in 6.0.0.*

# Configuration

### Build Time Configurations for MCUboot

The following build time configurations are defined in mcu-tools/include/mcuboot_config/mcuboot_config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Custom mcuboot_config.h | Manual Entry | | Add a path to your custom mcuboot_config.h file. It can be used to override some or all of the configurations |

| | | | |
|---|---|---|---|
| | | | defined here, and to define additional configurations. |
| Upgrade Mode | • Swap<br>• Overwrite Only<br>• Overwrite Only Fast<br>• Direct XIP | Overwrite Only | Swap supports A/B image swapping with rollback. Other modes with simpler code path, which only supports overwriting the existing image with the update image (Overwrite Only) or running the newest image directly from its flash partition (Direct XIP), are also available. |
| Validate Primary Image | • Enabled<br>• Disabled | Enabled | Always check the signature of the image in the primary slot before booting, even if no upgrade was performed. This is recommended if the boot time penalty is acceptable. |
| Downgrade Prevention (Overwrite Only) | • Enabled<br>• Disabled | Disabled | Prevent downgrades by enforcing incrementing version numbers. When this option is set, any upgrade must have greater major version or greater minor version with equal major version. This mechanism only protects against some attacks against version downgrades (for example, a JTAG could be used to write an older version). |
| Watchdog Feed | Manual Entry | | This function might be implemented if the OS / HW watchdog is enabled while doing a swap upgrade and the time it takes for a swapping is long enough to cause an unwanted reset. If implementing this, the OS main.c must also enable the watchdog (if |

| | | | |
|---|---|---|---|
| | | | required)! |
| Measured Boot | • Enabled<br>• Disabled | Disabled | Copies the boot data into the secure RAM, intended to be used by the secure App. |
| Data Sharing | • Enabled<br>• Disabled | Disabled | Copies the user data into the secure RAM, intended to be used by the secure App. |
| Signing and Encryption Options | | | |
| Signing and Encryption Options > TrustZone | | | |
| Boot Record (Image 2) | String length must be 12 characters or less. | | Create CBOR encoded boot record TLV for Image 2. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters] |
| Custom (Image 2) | Manual Entry | --confirm | Add any custom options to pass to imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent. |
| Signature Type | • None<br>• ECDSA P-256<br>• ECDSA P-384<br>• RSA 2048<br>• RSA 3072 | ECDSA P-256 | Configure the signature type. |
| Boot Record | String length must be 12 characters or less. | | Create CBOR encoded boot record TLV. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters] |
| Custom | Manual Entry | --confirm | Add any custom |

| | | | |
|---|---|---|---|
| | | | options to pass to imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent. |
| Python | Manual Entry | python | Name of the python command to use. Default is python, but can be updated to python3 for Linux or an absolute path if needed. |
| Encryption Scheme | • ECIES-P256<br>• RSA-OAEP (RSA 2048 only)<br>• Encryption Disabled | Encryption Disabled | Choose the encryption scheme. |
| Flash Configuration | | | |
| Dual Bank Mode | MCU Specific Options | | Enable dual bank mode for Direct XIP. |
| Memory Mirror Function (MMF) support | MCU Specific Options | | Use MMF for MCUBoot dxip mode |
| Data Sharing | | | |
| Maximum Measured Boot Record Size (Bytes) | Value must be an integer. | 0x64 | Maximum size of the boot record. |
| Shared Data Size (Bytes) | Value must be an integer. | 0x380 | Size of the shared RAM area. Required for Measured Boot. |
| Shared Data Address | Value must be an integer | 0x20000000 | Shared RAM start address. Required for Measured Boot. |

### Clock Configuration

- This module does not use peripheral clocks.
- For best performance it is recommended to use the fastest clock settings supported by the device in order to reduce boot times.
- **The bootloader must not disable the MOCO prior to calling the application. If the**

**bootloader disables the MOCO, then operation cannot be guaranteed.**

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Getting Started: Creating an MCUboot Project

Start by creating a new project in e² studio or RASC. Select 'Flat Project' template for single-image projects. For TrustZone based projects, select 'TrustZone Secure Project' template and set the clock circuit under the clocks tab to secure. On the Stacks tab, add New > Bootloader > MCUboot. Resolve any constraint errors and edit configurations as desired. Add either the example keys or generate your own key. The MCUboot key generation tool is provided at ra/mcu-tools/MCUboot/scripts/imgtool.py and documented at https://github.com/mcu-tools/mcuboot/blob/master/docs/imgtool.md. Install the following required python packages to use imgtool.py: https://github.com/mcu-tools/mcuboot/blob/master/scripts/requirements.txt.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot > Quick Setup > Call Quick Setup. Add a call to mcuboot_quick_setup() in the application and make any desired updates.

*Note*

*MCUboot will contain either the verification public key or its hash. During production it is necessary to permanently lock the flash region where MCUboot is programmed to prevent the keys or the code from being modified.*

### MCUboot Layout

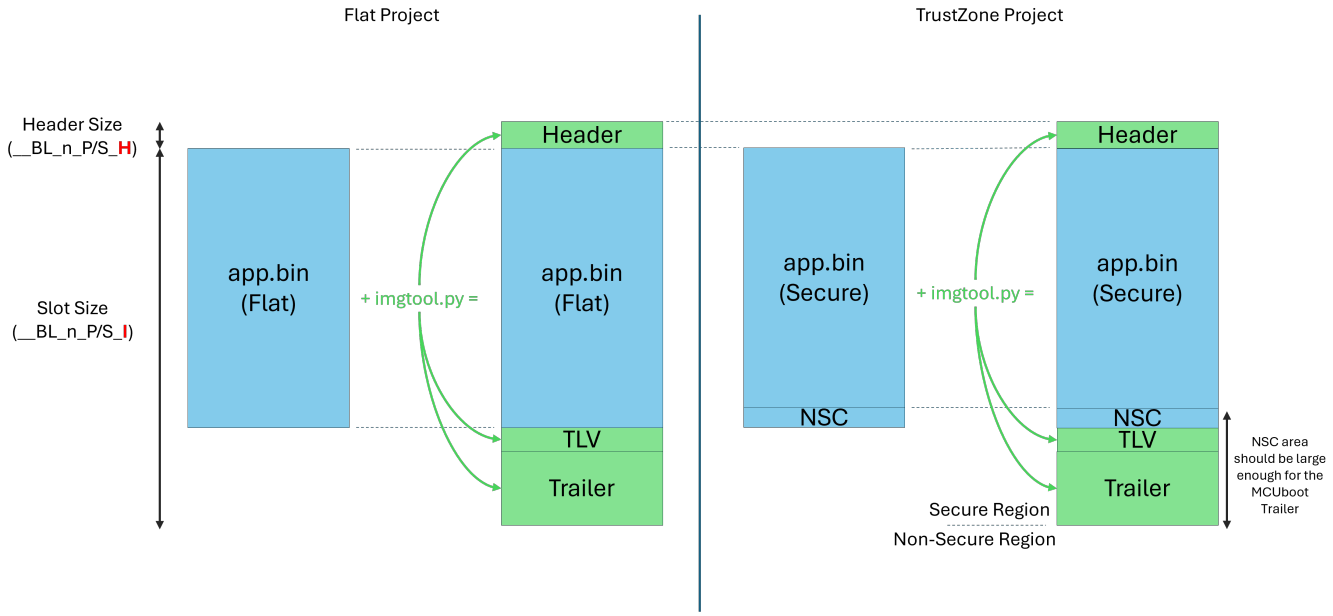



Figure 169: imgtool.py siging script adds header and trailer to user binary

### Getting Started: Signing Tool Prerequisite

To use the MCUboot signing tool, ensure you have Python 3.x installed on your system. Then install the Python packages required for the signing tool with the following command (modifying the path

as needed depending on current directory):

pip3 install --user -r ra/mcu-tools/MCUboot/scripts/requirements.txt

**Getting Started: Converting an Application Project to an MCUboot Image**

MCUboot application images must execute from the image slot defined by the MCUboot project. They are also limited to a single downloadable flash region. All of this is configured in the MCUboot module and the information is added to the generated SecureBundle (.sbd) file in the MCUboot project.

Any existing project can be converted to an MCUboot image.

1. If the project was created with a version prior to FSP v3.0.0, update the linker script to the v3.0.0 version before using it as an MCUboot application image.
2. Right click the project to convert in e² studio or RASC and select Properties.
3. Open C/C++ Build and select Build Variables.
4. Click Add...
5. For Variable Name, enter SmartBundle. For Type, select File. Browse to the *.sbd file created alongside the *.elf file for the associated MCUboot project. If the MCUboot project was specified as the Preceding Project during project creation, this step is done automatically.
6. Click OK, then Apply and Close.

To convert a TrustZone image, follow the steps above for the Secure project. The Non-Secure project will already use the SmartBundle from the Secure project which will inherit required data from the MCUboot SmartBundle.

MCUboot application images must also be signed to work with MCUboot. At a minimum, this involves adding a SHA and MCUboot specific constant data called boot magic in the image trailer. The size of the image trailer is (5 * BOOT_MAX_ALIGN) + (BOOT_MAX_IMG_SECTORS * min-write-size * 3).

Signing can be done on the as a post-build step in e² studio. To sign the image as a post-build step:

1. If Linux is used to develop the application image, change the MCUboot property Signing > Python to python3.
2. Build the bootloader project to generate the *.bld file. Make sure to build the bootloader project on the same computer as the application image to ensure the path to the signing script is correct.
3. Define environment variables in the Properties of the application image project in e² studio.
    a. Right click the application image project, and select **Properties**.
    b. Select **C/C++ Build > Environment** on the left.
    c. Click **Add...**
    d. Define the following environment variables one at a time:
        - MCUBOOT_IMAGE_VERSION: Set to the version of the application image.
        - MCUBOOT_IMAGE_SIGNING_KEY: Set the path to the key used for signing. If signing is not required, do not set this variable. If example keys are used, set MCUBOOT_IMAGE_SIGNING_KEY as follows (replace <boot_project> with the bootloader project path):
            - ECC 256: <boot_project>/ra/mcu-tools/MCUboot/root-ec-p256.pem
            - ECC 384: <boot_project>/ra/mcu-tools/MCUboot/root-ec-p384.pem
            - RSA 2K: <boot_project>/ra/mcu-tools/MCUboot/root-rsa-2048.pem

        - RSA 3K: <boot_project>/ra/mcu-tools/MCUboot/root-rsa-3072.pem
    - MCUBOOT_IMAGE_ENC_KEY: Set the path to the key used for encryption. If encryption is not required, do not set this variable. If example keys are used, set MCUBOOT_IMAGE_ENC_KEY as follows (replace <boot_project> with the bootloader project path):
        - ECIES: <boot_project>/ra/mcu-tools/MCUboot/enc-ec256-pub.pem
        - RSA 2K: <boot_project>/ra/mcu-tools/MCUboot/enc-rsa2048-pub.pem
    - MCUBOOT_APP_BIN_CONVERTER: Optional. Set to path to objcopy, arm-none-eabi-objcopy, fromelf, or ielftool. Not required if one of these tools is on the path.
4. Build the project.
5. The signed image is output next to the application <project>.elf file with the name <project>bin.signed.

**FSP 6.0.0 and newer**

An FSP Solution project must be created in order to set up the bootloader slots; the slot size and header information was previously in the MCUboot module.

## Bootloader region encoding in FSP 6.0.0

The header and image regions have to be named in the manner described below in the Solution project:

- FLASH_<core>_(B|S| ) - Indicates bootloader area or primary area. These are created by default and cannot be renamed.
- Examples:
    - FLASH_CM85_B: CM85 Bootloader area.
    - FLASH_CM85_S: CM85 Primary area.
    - FLASH_CM23 : CM23 Primary area.
- __BL: Indicates this is a bootloader symbol.
- _: delimiter between fields.
- ([0-9])|S: Image definitions field.
    - S indicates the Scratch area.
    - 0-9 correspond to traditional image IDs (only using 0/1 in legacy).
- _: delimiter between fields.
- (P|S): Indicates Primary or Secondary area.
    - S Secondary areas must have both Header (H) and Image (I) component.
    - P Primary areas must not have an I component (as the Image component is defined by the solution application region).
    - Secondary header must be the same size as Primary header.
    - Secondary images must be the same size as the area allocated for the project (Primary Image).
    - Headers must immediately precede the Image/application space.
- _: delimiter between fields
- (H|I): Indicates bootloader Header or Image component (not used with Scratch).
    - H indicates the MCUboot Header for Primary or Secondary (P/S) areas (use 0x200 for compatibility).
    - I indicates the MCUboot Secondary (S) Image allocation space (not used with Primary images).
- Primary or Secondary area size (including header and trailer) must be an integer multiple of the largest erase size on the mcu.

- Examples:
  - __BL_0_P_H: Image 0 Primary Header.
  - __BL_0_S_H: Image 0 Secondary Header.
  - __BL_1_S_I: Image 1 Secondary Image.
  - __BL_S : Scratch area.

1. Create the bootloader and set up any required configurations.
2. Build the bootloader project.
3. Create the application project, selecting the bootloader project as the "Preceding project" in the e2 studio wizard. The SmartBundle (.sbd) will be automatically linked.
4. Build the application project.
5. Create an "FSP Solution Project (Advanced)" and chose the final project in the chain (typically the application project).
6. Create the Header and Image regions according to the encoding described above.
7. Build the Solution project.

### Getting Started: Download and Debug

For projects that do not use TrustZone, debug the MCUboot project using the default configuration. Before running, load the signed image to the address specified in the signing comment in ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_config.h. This can be done with the Load Ancillary File button when debugging in e² studio. Upgrade images can be loaded to the upgrade image slots using the same method.

*Note*

*e² studio projects targeting RA8 devices that do not use TrustZone must disable the "Set TrustZone secure/non-secure boundaries" setting (Debug Configurations > Debugger > Connection Settings > TrustZone > Set TrustZone secure/non-secure boundaries).*

For TrustZone projects, debug using the Secure project to ensure the IDAU is partitioned correctly when debugging in e² studio. Make the following modifications before debugging in e² studio:

1. In the Debug Configurations for your project, on the Startup tab, click Add... to add the MCUboot project *.elf file (Image and Symbols), and optionally the Non-Secure project *.elf file.
2. For the Secure and Non-Secure project *.elf file, load Symbols Only.
3. After starting to debug, load the signed Secure image and the signed Non-Secure image into the addresses specified in the signing comment in ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_config.h. This can be done with the Load Ancillary File button when debugging in e² studio. Upgrade images can be loaded to the upgrade image slots using the same method.

### Confirming Upgrade in Swap Mode

In Swap Mode operation, if the upgrade image is signed with the –pad option, MCUboot will install that image as a temporary update where if nothing else is done, a reboot will cause MCUboot to revert to the image version that was swapped out during the upgrade. In order for the updated image to prevent this reversion and make the update permanent, the boot_set_confirmed() must be called from the application.

To avail this capability in the application image, from the Stacks tab, add New > Bootloader > MCUboot Image Utilities (Swap Mode). Resolve any constraint errors and edit configurations as desired.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot Image Utilities > Quick

Setup > Confirm Primary Image. Add a call to boot_set_confirmed() in the application and confirm the image in the primary slot.

### XIP Mode operation

XIP mode is enabled selecting "Direct XIP" as the Upgrade Mode option in the configurator or by defining "MCUBOOT_DIRECT_XIP" in the mcuboot config file. The linker script defines the symbol "XIP_SECONDARY_SLOT_IMAGE" by default to 0. To link an application to the secondar slot in XIP mode, set XIP_SECONDARY_SLOT_IMAGE to 1 in the application linker script. Direct XIP mode does not support TrustZone projects.

### XIP Mode with Memory Mirror Function (MMF)

*Note*

*Memory Mirror Functions is available on some MCUs, please check the hardware user manual for the availability. This mode only available in linear memory mode.*

When "Memory Mirror Function (MMF) support" under Common > Flash Layout is enabled, MCUBoot decides which image will be "mirrored". After the image is reflected on MMF region, the MCUBoot will start booting the application from start address of MMF.

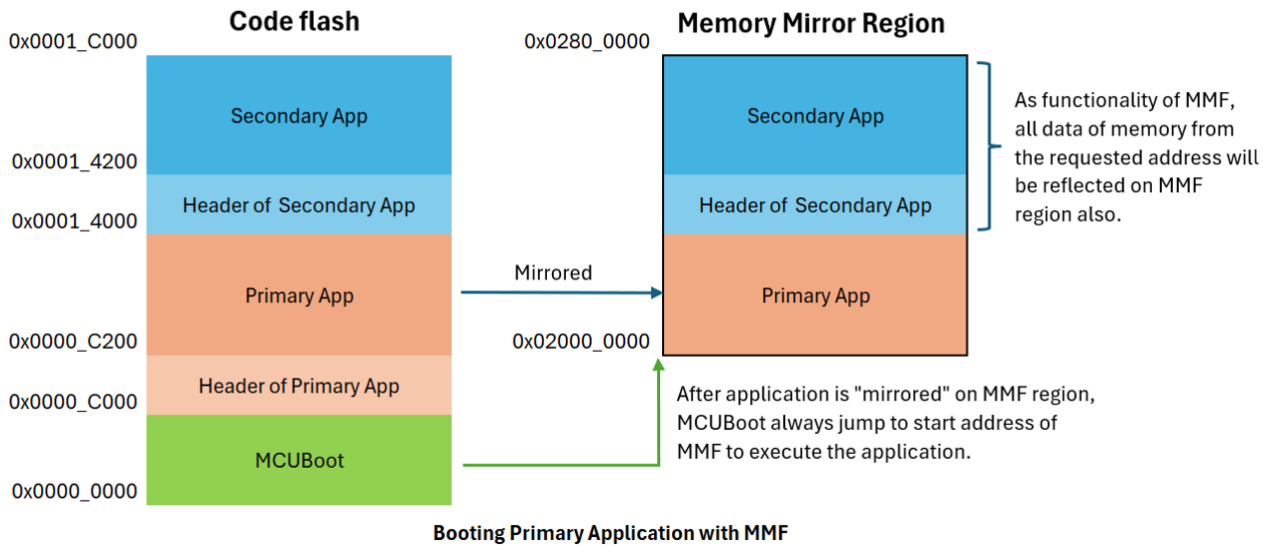



Figure 170: Booting Primary Application with MMF

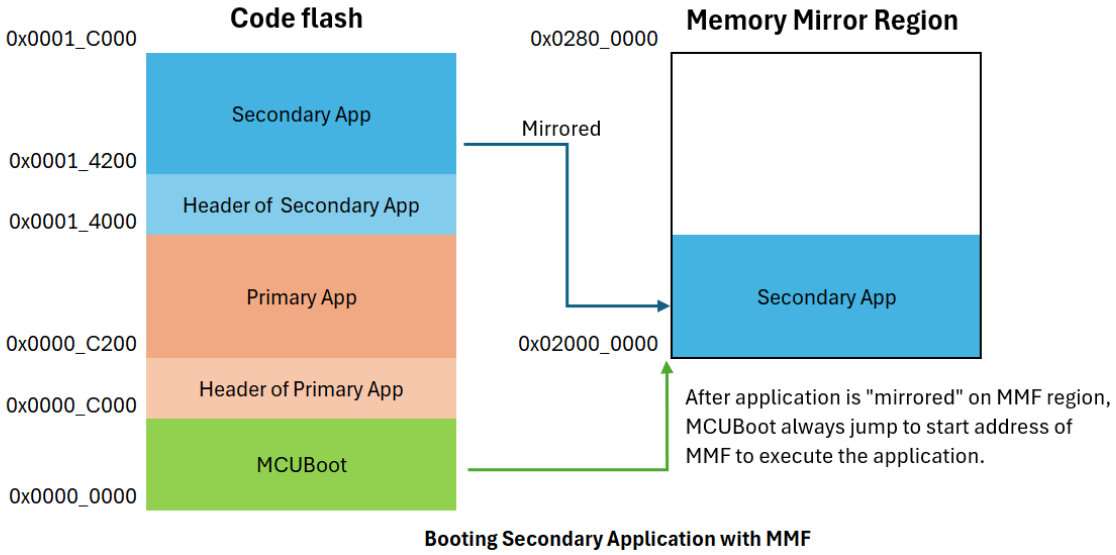



Figure 171: Booting Secondary Application with MMF

### Dual Bank operation

MCUboot can be used with Dual bank mode (with Flash HP or Flash LP) to leverage the advantages of dual bank flash operation. When Dual Bank mode is enabled, only the XIP upgrade mode can be used.

*Note*

*Unlike in normal XIP Mode operation, the linker script symbol "XIP_SECONDARY_SLOT_IMAGE" must be undefined in Dual Bank mode. An example flash layout in this configuration for a 1 MB is shown below. Note that there are 2 copies of the bootloader, one in Bank 0 and another in Bank 1.*

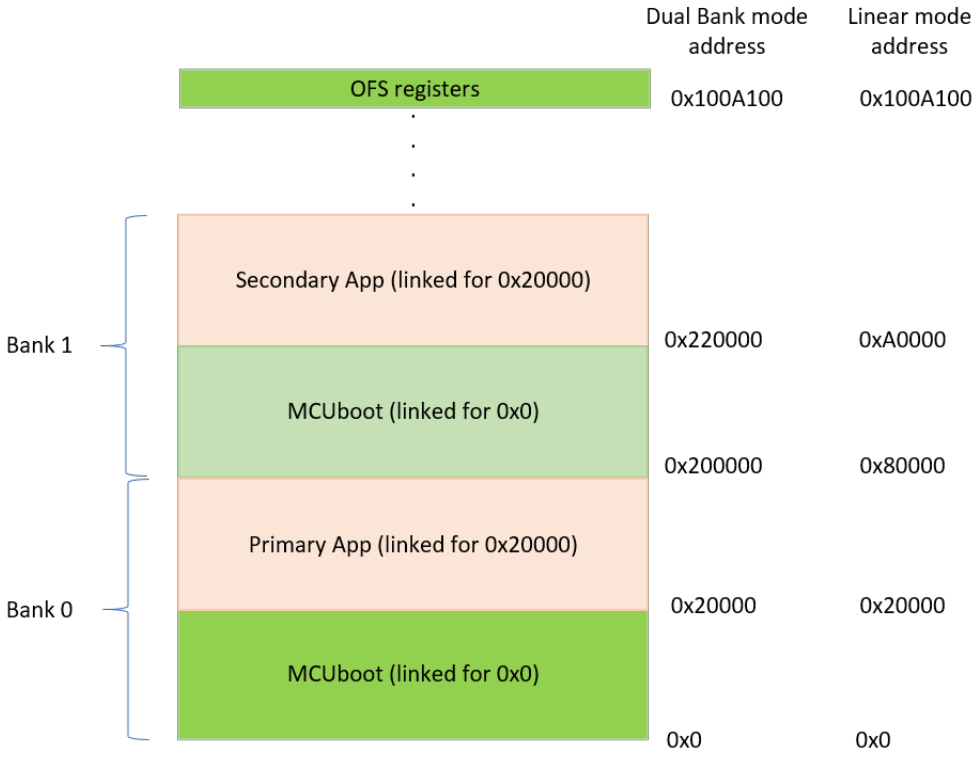



Figure 172: MCUboot Dual Bank layout of Flash HP for 1 MB Memory

For this example layout, the following files are generated when Dual Bank mode with XIP is enabled:

1. Bootloader Project: srec file linked to address 0 and includes the OFS region.
2. Application Project (Primary): signed bin file linked to address 0x20000.
3. Application Project (Secondary): signed bin file linked to address 0x20000.

With Flash HP in Dual Bank mode, the available flash memory is split into two halves and referred to as Bank 0 and 1. In this example Bank 0 would span from address 0x0 - 0x7FFFF and Bank 1 from 0x200000 - 0x27FFFF. In Linear mode, it is possible to program the Bank 1 area by programming to 0x80000 - 0xFFFFF.

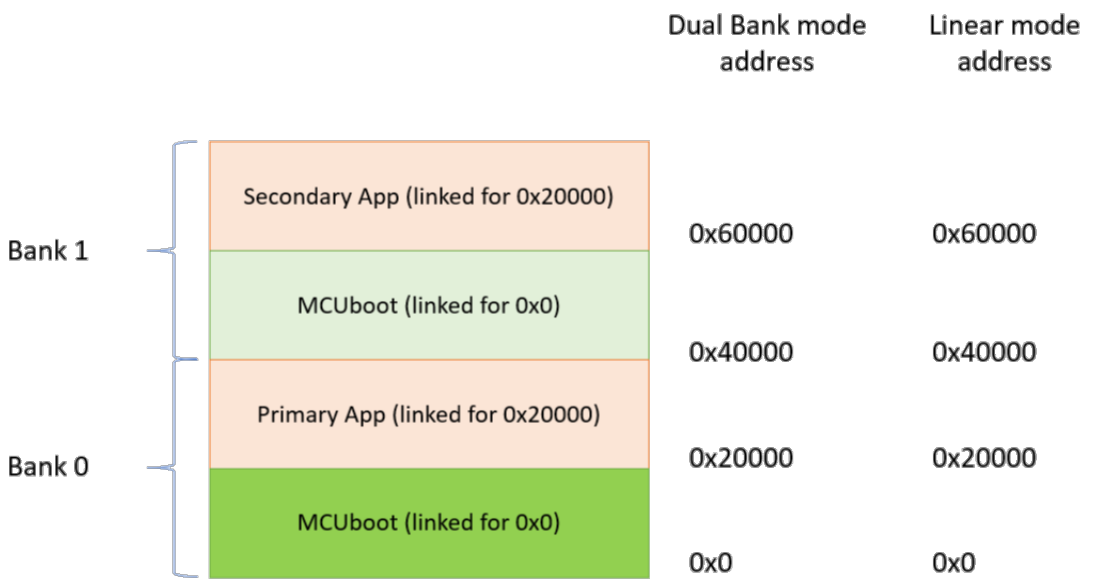



Figure 173: MCUboot Dual Bank layout of Flash LP for 512 KB Memory

With Flash LP in Dual Bank mode, the available flash memory is also split into two bank areas and referred to as Bank 0 and 1. In this example Bank 0 would span from address 0x0 - 0x3FFFF and Bank 1 from 0x40000 - 0x7FFFF. In Linear mode, it is possible to program the Bank 1 area by programming to 0x40000 - 0x7FFFF.

**Programming in Dual Bank Mode with Flash HP**

The bootloader must be duplicated in Dual Bank mode. Bank 0 can be programmed using the srec file generated from the MCUboot project; this will also program the OFS region which contains the dual bank enable bit, so it must either be programmed last if each file is programmed independently. Another option is to combine all images: MCUboot in Bank 0, primary image, MCUboot in Bank 1 (no OFS), and secondary image (optional). To program MCUboot to Bank 1, offset MCUboot by half the flash size and cut off the OFS region (0x0100A100 to address 0x0100A2FF on CM33 MCUs that support dual bank). Using srec-cat for a 1 MB flash MCU (0x80000 flash per bank), an example command to create the bootloader image for Bank 1: "srec_cat MCUboot_dualbank.srec -crop 0 0x80000 -offset 0x80000 -o MCUboot_dualbank_offset.srec". The application project for Bank 1 can be similarly offset using srec_cat: "srec_cat app1.bin.signed -binary -offset 0xA0000 -o app1_offset.srec", where 0xA0000 is 0x80000 (half the flash) + 0x20000 (MCUboot size). The signed binary file for Bank 0 can be converted to srec format using srec_cat: "srec_cat app0.bin.signed -binary -offset 0x20000 -o app0.srec"

To combine all the files into one srec file, use "srec_cat MCUboot_dualbank.srec MCUboot_dualbank_offset.srec app0.srec app1_offset.srec -o combined_srec".

**Programming in Dual Bank Mode with Flash LP**

To use Dual Bank Mode with Flash LP, enable Dual Bank Mode (Flash LP) in the MCUboot stack under Property > Common > General.

Similar to Dual Bank Mode with Flash HP, the bootloader must also be duplicated in Dual Bank mode. But the MCUBoot project does not need to be programmed last, because Flash LP does not use the

OFS region to enable/disable the dual bank mode.

### Dual bank booting without reset

*Note*

*This feature only available on MCUs which support flash_lp and Instant Bank Swap.*

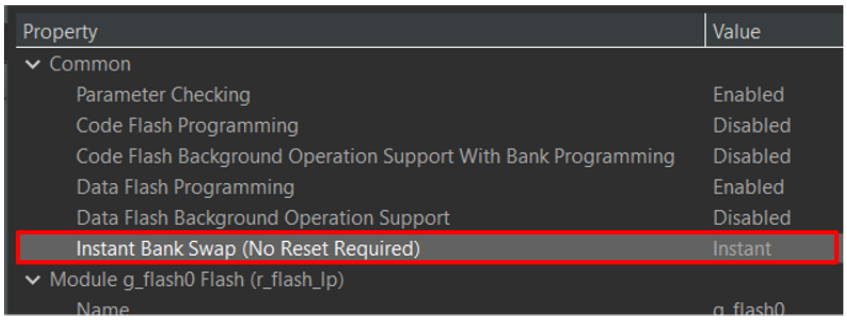



Figure 174: Instant Bank Swap option under Flash stack

The picture below shows the execution sequence with Reset and Without Reset (Instant Bank Swap).

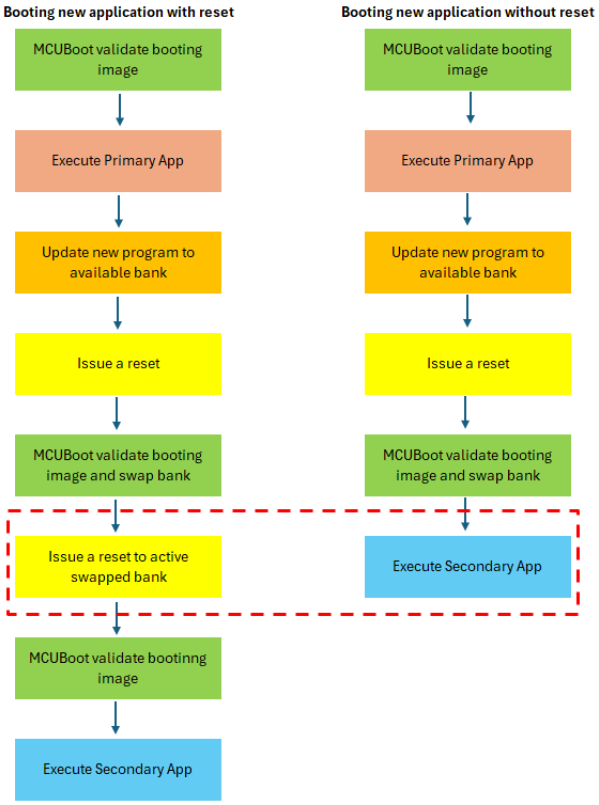



Figure 175: Software flow differences between booting with and without reset

### External Memory Support

# QSPI

QSPI support for secondary image storage can be enabled in the configurator. The bootloader expects the QSPI memory to be pre-configured by user code in the bootloader for read/write operation. The bootloader code operates under the assumption that the user has configured the QSPI in Extended-SPI mode and that R_QSPI_Open() has been called prior to invoking boot_go();. For example, on the EK_RA6M4 which has the MX25L25645G QSPI flash, after adding the QSPI module to the project, calling the following snippet will configure the QSPI for read/write operation:

```
/* Status Register (SREG) payload size */
# define SREG_SIZE 0x03
# define QSPI_COMMAND_WRITE_STATUS_REGISTER 0x01
# define QSPI_DEFAULT_SR1 0x40
# define QSPI_DEFAULT_SR2 0x00
/* Status register payload */
uint8_t   data_sreg[SREG_SIZE]                = {QSPI_COMMAND_WRITE_STATUS_REGISTER,
QSPI_DEFAULT_SR1, QSPI_DEFAULT_SR2};
R_QSPI_Open(&g_qspi0_ctrl, &g_qspi0_cfg);
/* Configure for Extended SPI Read/Write Mode */
R_QSPI_DirectWrite(&g_qspi0_ctrl, &(g_qspi0_cfg.write_enable_command), 1, false);
R_QSPI_DirectWrite(&g_qspi0_ctrl, data_sreg, SREG_SIZE, false);
```

For a more detailed example on how to initialize the QSPI device, refer to the QSPI module. The QSPI sector size must be the same as that of the MCU internal flash (BSP_FEATURE_FLASH_HP_CF_REGION1_BLOCK_SIZE) for swap mode operation.

# OSPI_B

OSPI_B support for secondary image storage can be enabled in the configurator for RA8: The bootloader expects the OSPI memory to be pre-configured by user code in the bootloader for read/write operation. The bootloader code operates under the assumption that the user has initialized the OSPI_B by calling R_OSPI_B_Open() prior to invoking boot_go();.

The OSPI sector size must be the same as that of the MCU internal flash (BSP_FEATURE_FLASH_HP_CF_REGION1_BLOCK_SIZE) for swap mode operation.

*Note*

*: The user can specify the OSPI erase block size for the area that will be erased/written to by MCUBoot. Do not use OSPI regions that span multiple block erase sizes or block erase sizes larger than the code flash erase size of 32K.*

***MCUboot Memory Map***

For single image projects with no external memory support, the default memory map looks like:

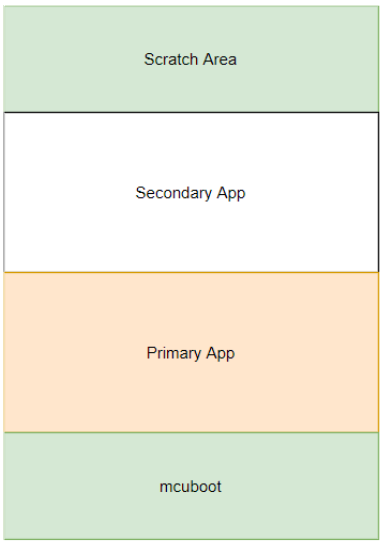



Figure 176: MCUboot Memory Map

For projects with 2 separately updateable images (used for TrustZone applications where the Secure and Non-Secure images can be updated separately), the default memory map with no external memory support looks like:

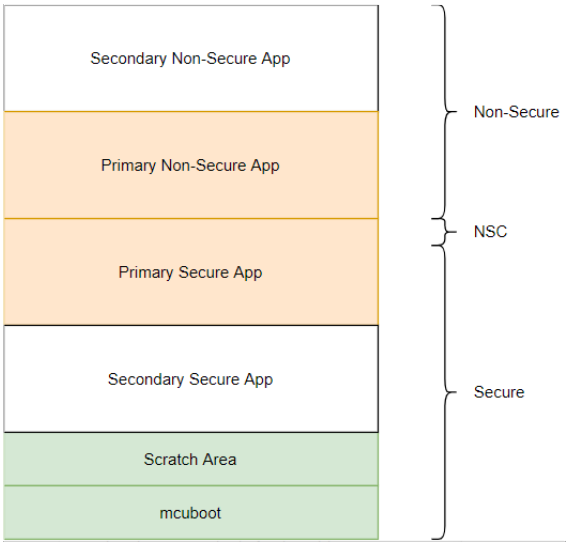



Figure 177: MCUboot Memory Map (TrustZone)

For single image projects with QSPI, the default memory map looks like:

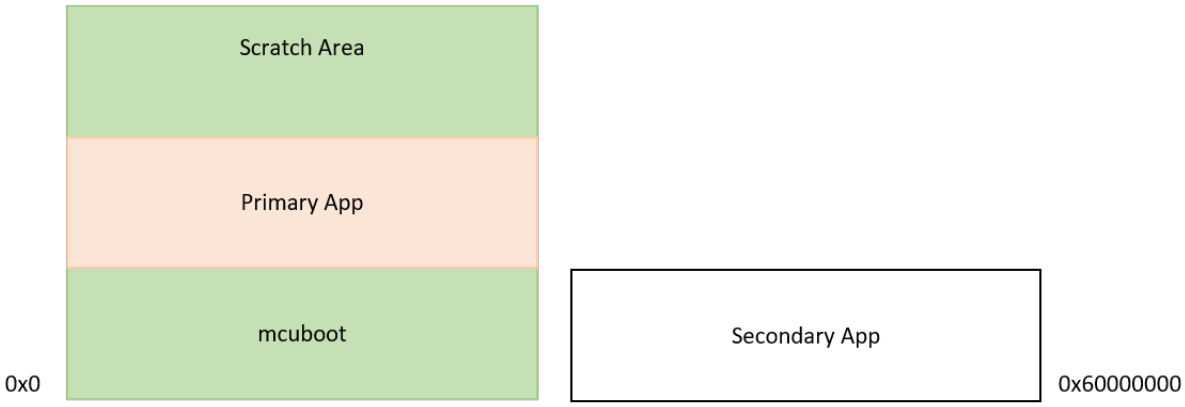



Figure 178: MCUboot Memory Map with QSPI

### MCUboot verification options

MCUboot in FSP supports the following secure image verification options

1. Hash verification only (SHA256).
2. Hash and signature verification (ECDSA-P256, ECDSA-P384, RSA-2048 and RSA-3072).
3. Hash, signature verification and image encryption (ECIES-P256 and RSA-OAEP-2048 with AES-128).
4. Hash and image encryption only.

MCUboot also supports signature verification using EdDSA-25519 and image encryption using AES-KW-128 and AES-KW-256 but those are currently not supported in FSP.

### Limitations

MCUboot swap mode is not functional with OSPI_B external storage.

### Notes

When encryption is enabled, MCUboot requires the image in the primary slot to be unencrypted. Only the image loaded in the secondary slot can be encrypted.

### MCUboot Crypto Stack Options

The following crypto stacks can be used with MCUboot in FSP:

1. MbedTLS, which is hardware accelerated on all RA devices. On the RA2 which has an AES engine only, ECC/RSA/SHA operations are in software.
2. TinyCrypt (S/W Only) can be used with all devices.
3. TinyCrypt (H/W Accelerated) has AES operations accelerated for the RA2 family only. When using MCUboot without encryption there is no difference between using this or the S/W only version.
4. SCE9 Protected Mode on devices that have the SCE9 (eg: RA6M4, RA4M3, RA4M2)

MbedTLS provides the best performance for MCUBoot signature verification on the RA6 and RA4 devices but has a much larger code footprint compared to TinyCrypt. For RA2 devices TinyCrypt is the best option.

### MCUboot boot time

The time from Reset to executing the application will depend on how quickly MCUboot finishes execution. This is dependent on a variety of factors including

1. The crypto algorithms chosen for image verification and whether hardware acceleration is enabled. Hardware accelerated SHA256 will be the fastest while encryption enabled modes will be the slowest.
2. The operating clock speeds.
3. Whether flash programming was required.

### Using SCE9 Protected Mode Crypto Stack

Using this crypto stack with MCUBoot provides additional security by ensuring that any keys that are used were securely provisioned for the specific device. The Application Note "Installing and Updating Secure Keys for RA Family" (R11AN0496) provides detailed steps on how to go about installing these keys. Since the section "Preparing Keys for Installation and Update Using RFP" document currently only provides information on how to install an AES key, this section will provide information on how to install an ECC public key. These steps can be used to install the public keys used for image verification or the keys used for image encryption.

*Note*

*When using the SCE9 Protected Mode Stack with MCUboot it is required that the public keys in the format described in the "MCUboot Example Keys" module in the stack is also provided in the project.*

### Installing public keys used for signature verification

1. Generate an ECC key pair. There are various ways to do this but you can use openSSL to do so: "openssl ecparam -name secp256k1 -genkey -noout -out my_ecc_secp256k1_key.pem".
2. Once the key is generated, in order to install the public key using RFP (Renesas Flash Programmer) the user needs to have their own UFPK (User Factory Programming Key) and W-UFPK (Wrapped User Factory Programming Key). Refer to R11AN0496 on how to obtain these keys.
3. Once the UFPK and W-UFPK are available, we need to extract the public key from the pem file. The public key can be viewed by using "openssl ec -noout -text -in my_ecc_secp256k1_key.pem". Note that when the ECC public key is printed out this way, it will contain a 0x04 ASN.1 prefix at the start, which should be discarded.
4. Use the rfp-util.exe utility from the RFP installation folder to wrap the public key using the UFPK and W-UFPK into a format that can be installed by RFP and the factory bootloader on the MCU.
5. Use RFP as described in R11AN0496 to install the key to the location of mcuboot_sce9_key section.

These are examples that install the default keys provided with MCUboot in ra/mcu-tools/MCUboot/. The examples assume that UFPK and W-UFPK are already available.

```
  //Print out the EC-P256 Public Key using openSSL
C:\ openssl ec -noout -text -in root-ec-p256.pem
read EC key
Private-Key: (256 bit)
priv:
    d7:98:d5:2f:83:01:24:3b:d3:54:2b:7e:55:ed:4c:
```

```
    74:61:19:00:b0:f9:50:5a:82:4f:e1:e8:ec:06:3b:
    cf:f1
pub:
    04:2a:cb:40:3c:e8:fe:ed:5b:a4:49:95:a1:a9:1d:
    ae:e8:db:be:19:37:cd:14:fb:2f:24:57:37:e5:95:
    39:88:d9:94:b9:d6:5a:eb:d7:cd:d5:30:8a:d6:fe:
    48:b2:4a:6a:81:0e:e5:f0:7d:8b:68:34:cc:3a:6a:
    fc:53:8e:fa:c1
ASN1 OID: prime256v1
NIST CURVE: P-256
//Use the public key (ignore the 0x04 ASN.1 prefix) in the RFP command line to
convert the public key into an installable format
C:\ "C:\Program Files (x86)\Renesas Electronics\Programming Tools\Renesas Flash
Programmer V3.08\rfp-util.exe" /genkey /ufpk "C:\ufpk.key" /wufpk
"C:\ufpk.key_enc.key" /key "2acb403ce8feed5ba44995a1a91daee8dbbe1937cd14fb2f245737e59
53988d994b9d65aebd7cdd5308ad6fe48b24a6a810ee5f07d8b6834cc3a6afc538efac1" /userkey
"16" /output "C:\ECC_pub_install.rkey"
// From the bootloader map file determine the address of mcuboot_sce9_key section
Use RFP to install "ECC_pub_install.rkey" as described in "Installing and Updating
Secure Keys for RA Family" (R11AN0496) to the address where the mcuboot_sce9_key
section is located.
```

```
  //Print out the RSA-2048 Public Key using openSSL
C:\ openssl asn1parse -in root-rsa-2048.pem
    0:d=0 hl=4 l=1187 cons: SEQUENCE
    4:d=1 hl=2 l=   1 prim: INTEGER         :00
    7:d=1 hl=4 l= 257 prim: INTEGER         :D106081A18442C18E8FBFDF70DA34F1FBBEE5EF
9AAD24B18D35AE96D188019F9F09C341BCBF3BC74DB42E78C7F10537E435E0D572C44D167080F0DBB5CEE
ECB399DFE04D840BAA774160ED152849A701B43C10E6698C2F5FAC414D9E5C14DFF2F8CF3D1E6FE75BBAB
4A9C8887E473C94C37767544BAA8D3835CA62617EB7E115DB7773D4BE7B7221896924FBF8656E643EC80E
D785D55C4AE4530D2FFFB7FDF31339833FA3AED20FA76A9DF9FEB8CEFA2ABEAFB8E0FA823754F43EE12BD
0D3085818F65E4CC8888131AD5FB08217F28A692723F3AB873E931A1DFEE8F81A246659F81CABDCCE681B
666435ECFA0D119DAF5C3AA7D167C647EFB14B2C62E1D1C9
  268:d=1 hl=2 l=   3 prim: INTEGER         :010001
```

```
  273:d=1 hl=4 l= 256 prim: INTEGER           :46A11421C52B5BFF3AD2D37924999745F0D9D62
BE505D42C5A56B0E39550CBF641D07667221E8502B38842F79D83E5C2977EF3610E6B5E9AC3055B2D8174
967505BCB96D57FE1D26D8E7A894EA9D209A99CD6624856BC22240F17C09D3B1960EE2F61BFFE9EE3277B
F4E539D9395FCA983F717EA4AFB2166E9FE2E0A25A87A9CAC06B57F8726B8FF42F68A9E9D9AF2963C9F8F
BA626886462342513EB249E64226609A9170C80A4F21CFE0BF27C10E26A0C26BEB36DF6D6C716F94C2ECA
559490F5429967E64BBB2FE991441EE06F17484D2BF91F6EE090A0F3B235BE01E19FC5894B1557D36ECE5
54AB91DF3B3836B244014DC131A61E55F41D9BA2737177C5
  533:d=1 hl=3 l= 129 prim: INTEGER           :FF7D8F5BE67045C74A5CC8FE4159315D477AC64
5F2EE64D73C4690B8E2D665082B4AA0051F5978E7FE9DD3DF4A009AE333A911F29297B73EF6208C8C1D33
E6DF7DA08729A32AF6849765CFF6C17E7FA16661445AAF8CBAF198777A2BCFF3AF90FD8CCE18AC2D452CE
204CCEB01FA085E0B701FCA7020F60B3F7CC1578E38F59B
  665:d=1 hl=3 l= 129 prim: INTEGER           :D170BF82EDAE6F933A1170678EAF4AD9DE0B953
C803EC21B01C312C2484E507E182454B6DAA5CB610500F7333E0D4C8B0AB97F3CA3761B41D056031BE032
1B787BFC917079A069B9CDF6D6A2B5AA3A899707702C23B5FC60332D7FF3EC655D9A9817A2A95B2498F25
0576D6A17278E707953F9D1EE45BD0ED2E9F774AF111E6B
  797:d=1 hl=3 l= 129 prim: INTEGER           :A43087E1D27CD28B19A1F955549FC256A4EA24E
3AD144160050F8050210F110CA7EDA45AC663D48C9B17C8A255C77FC2855FA0F617F9423D472571CD55B3
162B086C1290D29878A68B3955E5C941C739ED36931C0877536891C82E8E5B6CCAA64E1BA013410B32CA7
E52017301E9325965D65FC7D4398A857DFE69AE1FEB4103
  929:d=1 hl=3 l= 128 prim: INTEGER           :5043C1614FED75DD1A77EC78037AB258E47BD3E
9A7CC655F2C41B242BAAB28B5EA52A214A19EC05EA22848945EC781FA175617A9098C0DCE1F2597736B6C
4892D811673B8FA126638AC77A6248F4C01252CB0AF61F8972FAFB2208D356595292188F964B091EF16E8
BD3B59EDED8CE01D4BD96141A18A7E7B274EFDCCBEAE799
 1060:d=1 hl=3 l= 128 prim: INTEGER           :287481FC814FF38A3E47CABFE05BB7C1DCA3180
48FE4F21FEBD875E4DB3FD65227ACB9BAD31B3E0EF388B2DB27EF48712C0F0B0252988F88D0E9B8C44A78
65B942AF3006EE15AF1B634DCF14239A838EAC000DC93BC947F6A937E524400E16ACC48A118C819514BF8
51F549F3DF5BCC73693D45ED2E91685687F424B101837C9
//Use the public key in the RFP command line to convert the public key into an
installable format. Note that for RSA, the public modulus has to be concatenated to
the public exponent (typically 65537 in 32 bits 00010001 as shown in the asn1parse
output above) and then padded with 4 words of 0.
C:\ "C:\Program Files (x86)\Renesas Electronics\Programming Tools\Renesas Flash
Programmer V3.08\rfp-util.exe" /genkey /ufpk "C:\ufpk.key" /wufpk
"C:\ufpk.key_enc.key" /key "D106081A18442C18E8FBFDF70DA34F1FBBEE5EF9AAD24B18D35AE96D1
```

```
88019F9F09C341BCBF3BC74DB42E78C7F10537E435E0D572C44D167080F0DBB5CEEECB399DFE04D840BAA
774160ED152849A701B43C10E6698C2F5FAC414D9E5C14DFF2F8CF3D1E6FE75BBAB4A9C8887E473C94C37
767544BAA8D3835CA62617EB7E115DB7773D4BE7B7221896924FBF8656E643EC80ED785D55C4AE4530D2F
FFB7FDF31339833FA3AED20FA76A9DF9FEB8CEFA2ABEAFB8E0FA823754F43EE12BD0D3085818F65E4CC88
88131AD5FB08217F28A692723F3AB873E931A1DFEE8F81A246659F81CABDCCE681B666435ECFA0D119DAF
5C3AA7D167C647EFB14B2C62E1D1C90001000100000000000000000000000000" /userkey "0C" /output
"C:\RSA_pub_install.rkey"
// From the bootloader map file determine the address of mcuboot_sce9_key section
Use RFP to install "RSA_pub_install.rkey" as described in "Installing and Updating
Secure Keys for RA Family" (R11AN0496) to the address where the mcuboot_sce9_key
section is located.
```

```
  //Print out the RSA-3072 Public Key using openSSL
C:\ openssl asn1parse -in root-rsa-3072.pem
    0:d=0 hl=4 l=1764 cons: SEQUENCE
    4:d=1 hl=2 l=   1 prim: INTEGER          :00
    7:d=1 hl=4 l= 385 prim: INTEGER          :B42C0E985810A4A758997C01DD082A283433F89
61A34205D45C8712625E5D296EA7BB115AAA68A63228B2D4E8173BF6E15688C1AF4EF2A8F8C229E71574B
DE0F7E72D37AB8A71D44AD8700835CFD730572463F8BF91000D86ECC85EDF949DB783680493876DD5F540
4DA8C34A72B13256FD1154FADC2E1A5D24E570C7E9C9BBA4E68B2E02502AA00D3B4CC2F78E5BE47671FC8
6E226C5E61B69ACDE5A8BA7A80131B172E96EDCFB39BE41CE8ADA7F63A51665E998E87EE6025F88DBECEA
4A8CA936CD7BFD473338D4485CC7330089C4DB2AA5A6C6F7BABB7B37CC3FBE7CAC4F89A6FCBBB5B82E77A
E819FD2F1122FB7F768C6B94A4094FA56A7751EBA77EDA8706EEDCBED1EA1A401D1BFF1AB1517C12B0F3F
683019CE70C99BFAC685872A4B05985EE85AC2A22F4CF1508801F0DD01EA0A094C8F7FA65DD52E8963723
305736E69DF40C4A05751FAD01CAB76D8C4374060A81F30162FFF7F55FAFE72B0EF881B565DD01D99F071
78A18CF236E886591B57BD3B02DAF93666374AC5AE673DE3B
  396:d=1 hl=2 l=   3 prim: INTEGER          :010001
  401:d=1 hl=4 l= 385 prim: INTEGER          :98C34E30AF629528EABF605C781B671B257FF74
2D5BEE2BE12DFEBC80B93FC6547354F256EC6BC4967CD97C19B931779701F6FC39F6F75A7B68AD7CA83D8
E8D43C4381B9E8FC909D5D803CD824AD24AC3683077857D9D0CDB1CC29B6678ACED1F36BFC292AE771DF5
C2A2D7CAB4CA374378590CB392A2686A17518EB9822930B7955DE6C9C14D12DD852D05963E96FD73CC1BF
005AA185B8D5CA15CFA6AB4E186F9AA5A2340838F631B44D2A9FAAECE3EF869BA9192779DEFDF2EBC41F9
38F249EABFFB1EB8DB49A9F61DF86242059E6D1E8D579C6D980CBFFF68AE36AAE6F5ED50AE29E3A35B5E1
CFEB78731906E4F47CD7748CCC9B0F37BC73A9A459AD78E092C4923127687846 5B15967FFB3B93788C085
```

```
DA6ABBFE405819561DE7C4B16BF17C653098176DC45D0CE29F99E6A188217FD6187C7E120F6AB0FD4D867
32143C0C98C23CDCDE8FB766B221DC202B08FD227CE176F2E4A946F0E25BB184A9CF2778FCF0FBF3D5574
932BF07020DDC866C5A1370D46EC039FCDC87CE9F5CDA42A1
  790:d=1 hl=3 l= 193 prim: INTEGER          :E5BF240823A33C7AD9D931818534A0F1A5C45A8
6B178CE710D5B7203B26E35D38E28CBEAAFF1F47B34523EF3CC5C6E6FF3CD0FA8CF814601C4CD7D46B524
0208FE1D68271ED0780172B651AAE357319C67D51927A69A910E819AB35959DE16767CB3FF35B2ADF9FCE
CE943EA4A81337C498823C9241B7F9A491021D5DE8697EA0FC365C114E899E0616C86E20028BC1A61FC45
DA3E7DEC687528EEEEF9C83E879CBC5CC5AE02FFF66CF4D3879910F15B5005ABA73DAAA1E30308B0A16CF
FD4F1
  986:d=1 hl=3 l= 193 prim: INTEGER          :C8C2B1D568DC756EC05384E6328598EDDC1C822
9E10731277BBCF98609DA6EC9C56AE751A408B1FC8053D06D3F9431015E36BE8FC579E89CBBC4254B9898
B4E389E37F30A7D1B6ED387384DC1CED18FEDE59138337EDAEFD5F7193ABFBCB847B34E2F313BFB5FD2BA
9698596E6E428F59D6F09A994ECCB885EAE9BF414B71C3F2F4BD22978340824A43D1B434844DF6768BC32
7CB934117043A84D159575AEB038BC07D5B9650D25F5D6365CB0B1F06C7B16873C500F3912F8AFB5FA42D
DB5EB
 1182:d=1 hl=3 l= 192 prim: INTEGER          :0B2C1F6171F23737B62E54B4FAB853774CDB5E9
79C3BC6B642B306B95D4CF4BB23F7A1EAF8C1686531672A7E8A9A9818439FDE3A1455320555FFB42EE312
2C3329C05E5740AD5C989D6E764C3C1FFA5EA3C1FE262A78EBC2EBD48D123447938A11886349C635A5A98
20FD9E8A3D4E29F11A8582D28DF597634455F8FD9F16BA3CB3E724D5069FB49A91330FFFE87FC95BFDC17
BD843A756BA2FEE9979F77F86A79D9C31D2DD82180674E04975C7F316D8257B4403EC47478E8A5DC890D1
6E7D1
 1377:d=1 hl=3 l= 192 prim: INTEGER          :543B63F3BF6868190CB6BE16FB7194459049A1C
F426C0B129ED71DF6402216C3AF81F8060805E1EFA844023A2427E01BCBD4BA45863C6CFD7DC681436386
06B7453E5F3A21DF5A99D34A9C9EE1C014F1B286BB2A1E082A9882381C1657B1FF26D67CB6323E08746DA
249F4D3E89228214D69AE2B29A1E48F95F23ADAC0EA46FEB7B05F4028FFE3BEC3EE23872A46435996D707
73CF1CEA8828CAEF74B3DDC96A849357D23354139D2EB52EDFDAFEFD79F676F04CBCE67632E086909AACF
D6AFB
 1572:d=1 hl=3 l= 193 prim: INTEGER          :BD64D5B1F4FB1B37B0C423C57CA8E4261EE5F9A
2D72F069EDD39922BE0BC66863CB21BDA51BC2517C885AB0ED380EBBF19D6E440626A9EACAED005F8C539
E7F1235E7F7B7ECD53BC3470676FD22510800877A91675EB3CCCF574105EF081BBB3C022E564C3B956DBE
D3E3FF1FC12AEA027205EE84301C5C25A439C519424B64227FF87FE38DC9F64E04E718810D4F09522287E
17164FE3AD1AD94B2D4E09EBE974EEF55FB9E1287E824C7EA5A30D4A4EAF71669CFF8AD9DAD95859F3BFF
DD182
//Use the public key in the RFP command line to convert the public key into an
```

```
installable format. Note that for RSA, the public modulus has to be concatenated to
the public exponent (typically 65537 in 32 bits 00010001 as shown in the asn1parse
output above) and then padded with 4 words of 0.
C:\ "C:\Program Files (x86)\Renesas Electronics\Programming Tools\Renesas Flash
Programmer V3.08\rfp-util.exe" /genkey /ufpk "C:\ufpk.key" /wufpk
"C:\ufpk.key_enc.key" /key "B42C0E985810A4A758997C01DD082A283433F8961A34205D45C871262
5E5D296EA7BB115AAA68A63228B2D4E8173BF6E15688C1AF4EF2A8F8C229E71574BDE0F7E72D37AB8A71D
44AD8700835CFD730572463F8BF91000D86ECC85EDF949DB783680493876DD5F5404DA8C34A72B13256FD
1154FADC2E1A5D24E570C7E9C9BBA4E68B2E02502AA00D3B4CC2F78E5BE47671FC86E226C5E61B69ACDE5
A8BA7A80131B172E96EDCFB39BE41CE8ADA7F63A51665E998E87EE6025F88DBECEA4A8CA936CD7BFD4733
38D4485CC7330089C4DB2AA5A6C6F7BABB7B37CC3FBE7CAC4F89A6FCBBB5B82E77AE819FD2F1122FB7F76
8C6B94A4094FA56A7751EBA77EDA8706EEDCBED1EA1A401D1BFF1AB1517C12B0F3F683019CE70C99BFAC6
85872A4B05985EE85AC2A22F4CF1508801F0DD01EA0A094C8F7FA65DD52E8963723305736E69DF40C4A05
751FAD01CAB76D8C4374060A81F30162FFF7F55FAFE72B0EF881B565DD01D99F07178A18CF236E886591B
57BD3B02DAF93666374AC5AE673DE3B0001000100000000000000000000000" /userkey "0E"
/output "C:\RSA_3072_pub_install.rkey"
// From the bootloader map file determine the address of mcuboot_sce9_key section
Use RFP to install "RSA_3072_pub_install.rkey" as described in "Installing and
Updating Secure Keys for RA Family" (R11AN0496) to the address where the
mcuboot_sce9_key section is located.
```

# Examples

### Basic Example

This is an example of using MCUboot in an application.

```
void rm_mcuboot_port_example (void)
{
#ifdef MCUBOOT_USE_MBED_TLS
 /* Initialize mbedtls. */
 mbedtls_platform_context ctx = {0};
    assert(0 == mbedtls_platform_setup(&ctx));
#elif defined(MCUBOOT_USE_OCRYPTO)
 /* Initialize TinyCrypt port. */
```

```
    assert(FSP_SUCCESS == RM_OCRYPTO_PORT_Init());
#elif defined(MCUBOOT_USE_TINYCRYPT)
 /* Initialize TinyCrypt port. */
    assert(FSP_SUCCESS == RM_TINCYRYPT_PORT_Init());
#else
 /* Initialize SCE9 Protected Mode driver. */
 sce_instance_ctrl_t sce_ctrl;
 const sce_cfg_t    sce_cfg =
    {.lifecycle = SCE_SSD};
    assert(FSP_SUCCESS == R_SCE_Open(&sce_ctrl, &sce_cfg));
#endif
 /* (Optional) To check for updates, call boot_set_pending. */
 bool update = 0;
 if (update)
    {
       boot_set_pending(0);
    }
 /* Verify the boot image and get its location. */
 struct boot_rsp rsp;
    assert(0 == boot_go(&rsp));
 /* Enter the application. */
    RM_MCUBOOT_PORT_BootApp(&rsp);
}
```

## 5.2.5 CapTouch

Modules

### Detailed Description

CapTouch Modules.

### Modules

| | |
|---|---|
| | CTSU (r_ctsu)<br>This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It |

| | |
|---|---|
| | implements the CTSU Interface. |
| | Touch (rm_touch)<br><br>This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch Middleware Interface. |

### 5.2.5.1 CTSU (r_ctsu)

Modules » CapTouch

**Functions**

| | |
|---|---|
| fsp_err_t | R_CTSU_Open (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)<br><br>Opens and configures the CTSU driver module. Implements ctsu_api_t::open. More... |
| fsp_err_t | R_CTSU_ScanStart (ctsu_ctrl_t *const p_ctrl)<br><br>This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart. More... |
| fsp_err_t | R_CTSU_DataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)<br><br>This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet. More... |
| fsp_err_t | R_CTSU_AutoJudgementDataGet (ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status)<br><br>This function gets the result of automatic judgement button. Call after the scan is completed. The result is 64-bit bitmaps and is stored in order of TS number for specified ctsu control. After the initial judgement, the baseline initialization bit is set and the automatic judgement threshold is set. This function is called only for automatic judgement. Implements ctsu_api_t::autoJudgementDataGet. More... |
| fsp_err_t | R_CTSU_OffsetTuning (ctsu_ctrl_t *const p_ctrl)<br><br>This function tunes the offset register(SO). Call after the |

| | |
|---|---|
| | measurement is completed. If the return value is FSP_ERR_CTSU_INCOMPLETE_TUNING, tuning is not complete. Execute the measurement and this function call routine until the return value becomes FSP_SUCCESS. It is recommended to run this routine after R_CTSU_Open(). It can be recalled and tuned again. When the automatic judgement is enabled, after the offset tuning is completed,the baseline initialization bit flag is set. Implements ctsu_api_t::offsetTuning. More... |
| fsp_err_t | R_CTSU_ScanStop (ctsu_ctrl_t *const p_ctrl)<br>This function scan stops the sensor as scanning by the CTSU. Implements ctsu_api_t::scanStop. More... |
| fsp_err_t | R_CTSU_CallbackSet (ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void *const p_context, ctsu_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_CTSU_Close (ctsu_ctrl_t *const p_ctrl)<br>Disables specified CTSU control block. Implements ctsu_api_t::close. More... |
| fsp_err_t | R_CTSU_SpecificDataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type)<br>This function gets the sensor specific data values as scanned by the CTSU. Call this function after calling the R_CTSU_DataGet() function. More... |
| fsp_err_t | R_CTSU_DataInsert (ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data)<br>This function inserts the value of the second argument as the measurement result value. Call this function after calling the R_CTSU_DataInsert() function. Implements ctsu_api_t::dataInsert. More... |
| fsp_err_t | R_CTSU_Diagnosis (ctsu_ctrl_t *const p_ctrl)<br>Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis. More... |

### Detailed Description

This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It implements the CTSU Interface .

# Overview

The capacitive touch sensing unit HAL driver (r_ctsu) provides an API to control the CTSU peripheral. This module performs capacitance measurement based on various settings defined by the configuration. This module is configured via the QE for Capacitive Touch.

### Features

- Supports multiple scan modes
  - Self-capacitance multi scan mode (CTSU2 support active shield)
  - Mutual-capacitance full scan mode
  - Mutual-capacitance parallel scan mode (CTSU2)
  - Current Measurement mode (CTSU2)
  - Diagnosis scan mode
- Scans may be started by software or an external trigger
- Returns measured capacitance data on scan completion
- Support DTC transfer of scanned data
- Supports TrustZone
- Corrects accuracy for temperature drift (CTSU2)

# Configuration

*Note*

***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e² studio and search for "QE".***

### Build Time Configurations for r_ctsu

The following build time configurations are defined in fsp_cfg/r_ctsu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support for using DTC | • Enabled<br>• Disabled | Disabled | Enable DTC support for the CTSU module. |
| Enable automatic judgment mode | • Enabled<br>• Disabled | Disabled | Enable or Disable automatic judgment mode. This feature is only for RA4L1 and RA0L1. |
| Interrupt priority level | MCU Specific Options | | Priority level of all CTSU interrupt (CSTU_WR,CTSU_RD,CTSU_FN) |

### Configurations for CapTouch > CTSU (r_ctsu)

This module can be added to the Stacks tab via New Stack > CapTouch > CTSU (r_ctsu). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Scan Start Trigger | MCU Specific Options | | CTSU Scan Start Trigger Select |

**Interrupt Configuration**

The first R_CTSU_Open function call sets CTSU peripheral interrupts. The user should provide a callback function to be invoked at the end of the CTSU scan sequence. The callback argument will contain information about the scan status.

**Clock Configuration**

The CTSU peripheral module uses PCLKB as its clock source. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

*Note*

*The CTSU Drive pulse will be calculated and set by the tooling depending on the selected transfer rate.*

**Pin Configuration**

The TSn pins are sensor pins for the CTSU.

The TSCAP pin is used for an internal low-pass filter and must be connected to an external decoupling capacitor.

# Usage Notes

The CTSU module is a CTSU driver for the Touch module. The CTSU module assumes the access from the Touch middleware layer, and it is also accessible from an user application.
The CTSU peripheral installed on RA family has three versions: CTSU, CTSU2, and CTSU2SLa. Each MCU devices are equipped with the following version of CTSU peripherals.

- CTSU2SLa : RA4L1, RA0L1
- CTSU2 : RA2E1, RA2L1
- CTSU : RA2A1, RA4M1, RA4M2, RA4M3, RA4W1, RA6M1, RA6M2, RA6M3, RA6M4, RA6M5,

CTSU, CTSU2 and CTSU2SLa are functionally different, so CTSU, CTSU2 and CTSU2SLa are described in this application note as below.
Common description for CTSU and CTSU2 -> CTSU
Description only for CTSU -> CTSU1
Description only for CTSU2 and CTSU2SLa -> CTSU2
Description only for CTSU2SLa -> CTSU2SLa
Without mention, it means the common description for CTSU, CTSU2 and CTSU2SLa.

**Functions**

The CTSU module supports the following functions.

**Touch judgement Type**

CTSU2 has two judgement type, which differ in processing and output data.
It is common to get three raw value for each element and calculate CCO correction, and then performs different processing.

1. Value Majority Mode(VMM)
   Calculate three frequency correction value from three CCO correction values. Calculate the sum of the two closest values of the three frequency correction value. Outputs the moving average of this added value.

2. judgement Majority Mode (JMM)
   This mode is supported from FSP V5.6.0 or later.
   Outputs the moving average of three CCO correction values.

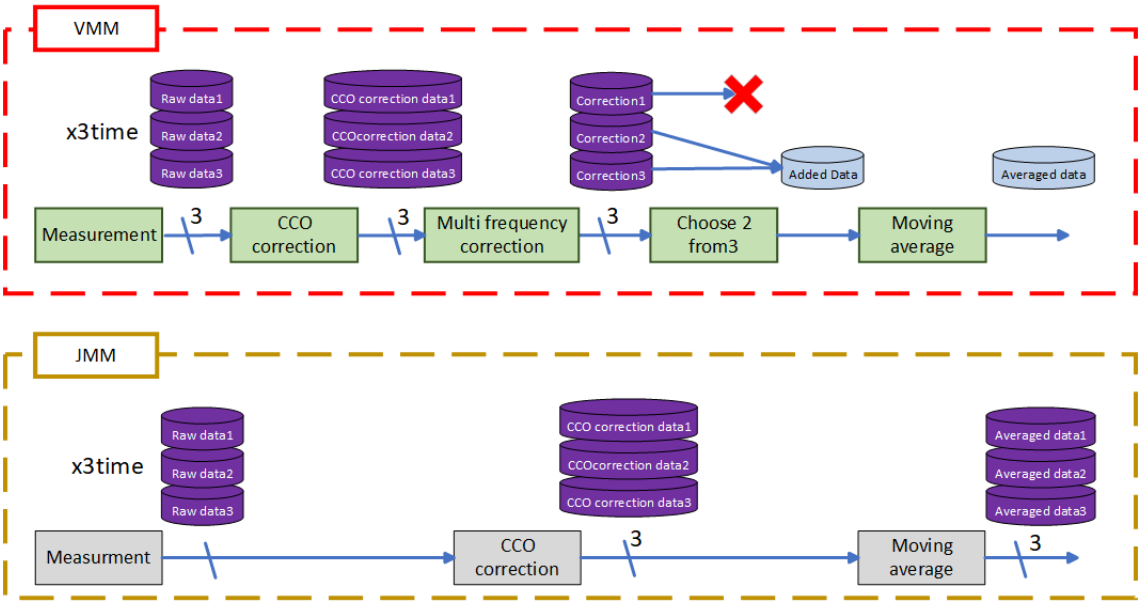



Figure 179: Data flow Comparison of JMM and VMM

### Measurements and Obtaining Data

Measurements can be started by a software trigger or by an external event triggered by the Event Link Controller (ELC).
As the measurement process is carried out by the CTSU2 peripheral, it does not use up main processor processing time.
The CTSU module processes INTCTSUWR and INTCTSURD if generated during a measurement. The data transfer controller (DTC) can also be used for these processes.
When the measurement complete interrupt (INTCTSUFN) process is complete, the application is notified in a callback function. Make sure you obtain the measurement results before the next measurement is started as internal processes are also executed when a measurement is completed.
Start the measurement with API function R_CTSU_ScanStart().
Obtain the measurement results with API function R_CTSU_DataGet().
The number of arrays in the second argument of R_CTSU_DataGet() is different between VMM and JMM.
VMM returns one measurement result for each element.
JMM returns three measurement results for each element.

### Sensor CCO Correction function

The CTSU2 peripheral has a built-in correction circuit to handle the potential microvariations related to the manufacturing process of the sensor CCO MCU.
The module temporarily transitions to the correction process during initialization after power is turned on. In the correction process, the correction circuit is used to generate a correction coefficient (factor) to ensure accurate sensor measurement values.
When temperature correction is enabled, an external resistor connected to a TS terminal is used to periodically update the correction coefficient. By using an external resistor that is not dependent on temperature, you can even correct the temperature drift of the sensor CCO.

### Initial Offset Adjustment

The CTSU2 peripheral was designed with a built-in offset current circuit in consideration of the amount of change in current due to touch. The offset current circuit cancels enough of the parasitic capacitance for it to fit within the sensor CCO dynamic range.
This module automatically adjusts the offset current setting. As the adjustment uses the normal measurement process, the combination of R_CTSU_ScanStart() and R_CTSU_DataGet() or the combination of R_CTSU_ScanStart() and R_CTSU_OffsetTuning() must be repeated several times after startup. Because the ctsu_element_cfg_t member "so" is the starting point for adjustments, you can set the appropriate value for "so" in order to reduce the number of times the two functions must be run to complete the adjustment. Normally, the value used for "so" is a value adjusted by QE for Capacitive Touch.
R_CTSU_OffsetTuning() was added in FSP 3.8.0. This API can also be used for initial offset adjustment, and offset adjustment can be performed again at any time. See example code of R_CTSU_OffsetTuning() for details.

| Mode | CTSU1 target value | CTSU2 target value |
|---|---|---|
| Self-capacitance | 15360 (37.5%) | 11520 (37.5%) |
| Self-capacitance using active shield | - | 4608 (15%) |
| Mutual-capacitance | 10240 (25%) | 7680 (25%) |

The percentage is based on 100% being the maximum input current applied to the CCO.
CTSU1: 100% is the measured value 40960 when the measurement time is 526us(base time).
CTSU2: 100% is the measured value 30720 when the measurement time is 256us(base time).
When the measurement time is changed, the target value is adjusted by the ratio with the base time.

Example of target value in combination of CTSUSNUM and CTSUSDPA.
CTSU1 (CTSU clock = 32MHz, Self-capacitance mode)

| Target value | CTSUSNUM | CTSUSDPA | Measurement time |
|---|---|---|---|
| 15360 | 0x3 | 0x7 | 526usec |
| 30720 | 0x7 | 0x7 | 1052usec |
| 30720 | 0x3 | 0xF | 1052usec |
| 7680 | 0x1 | 0x7 | 263usec |

| | | | |
|---|---|---|---|
| 7680 | 0x3 | 0x3 | 263usec |

The measurement time changes depending on the combination of CTSUSNUM and CTSUSDPA. When changing CTSUPRRATIO and CTSUPRMODE from the recommended values, follow the Hardware Manual for the measurement time.

CTSU2 (Self-capacitance mode)

| Target value | Target value (multi frequency) | CTSUSNUM | Measurement time |
|---|---|---|---|
| 5760 | 11520 (128us + 128us) | 0x7 | 128usec |
| 11520 | 23040 (256us + 256us) | 0xF | 256usec |
| 2880 | 5460 (64us + 64us) | 0x3 | 64usec |

The measurement time changes depending on CTSUSNUM. If STCLK cannot be set to 0.5MHz, it will not support the table above. Regarding STCLK, refer to the hardware manual.

### Random Pulse Frequency Measurement (CTSU1)

The CTSU1 peripheral measures at one drive frequency.
The drive frequency determines the amperage to the electrode and generally uses the value tuned with QE for Capacitive Touch.
The drive frequency is calculated as below.
It is determined by PCLK frequency input to CTSU, CTSU Count Source Select bit(CTSUCLK), and CTSU Sensor Drive pulse Division Control bit(CTSUSDPA). For example, If it is set PCLK =32MHz, CTSUCLK = PLCK/2, and CTSUSDPA = 1/16, then drive frequency is 0.5MHz. CTSUSDPA can change for each TS port.

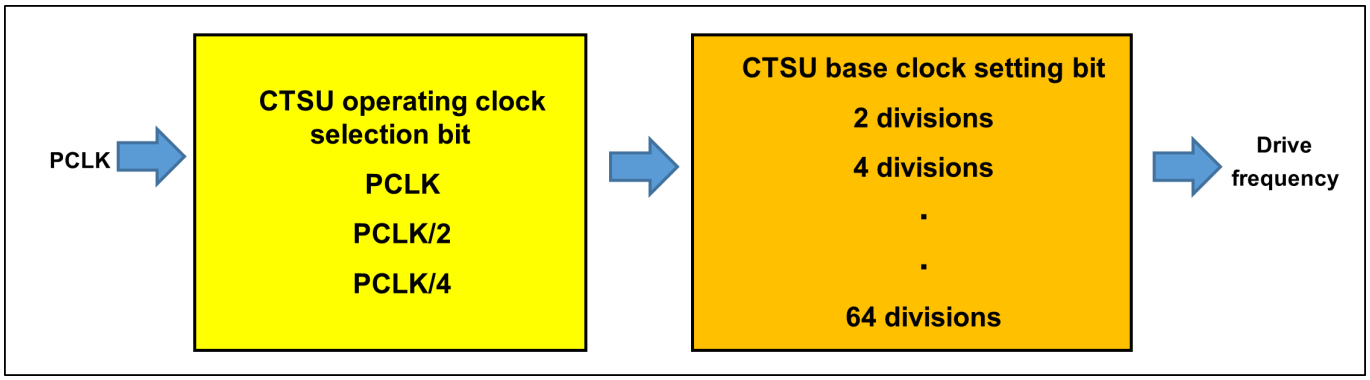



Figure 180: Drive Frequency Settings

The actual drive pulse is phase-shifted and frequency-spread with respect to the clock based on the drive frequency as a measure against external environmental noise. This module is fixed at initialization and sets the following.
CTSUSOFF = 0,CTSUSSMOD = 0,CTSUSSCNT = 3

### Multi-frequency Measurements (CTSU2)

The CTSU2 peripheral can measure in one of four drive frequencies to avoid synchronous noise.

With the default settings, the module takes measurements at three different frequencies. This data are called raw data. And calculate CCO correction. This data are called CCO correction data.
In VMM, After standardizing the results obtained at the three frequencies in accordance with the first frequency reference value, the measured value is determined based on majority in a process referred to as "normalization."
The three values standardized to the first frequency reference value are called frequency correction data.
You can get the three correction data with R_CTSU_SpecificDataGet().
In VMM, three raw data, CCO correction data, and frequency correction data can be obtained with R_CTSU_SpecificDataGet().
In JMM, three raw data and three CCO correction data can be obtained with R_CTSU_SpecificDataGet().

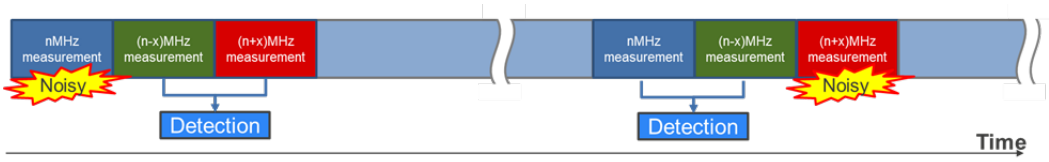



Figure 181: Multi-frequency Measurements

Drive frequency is determined based on the config settings. The module sets registers according to the config settings, and sets the three drive frequencies.
Drive frequency is calculated in the following equation:
(PCLKB frequency / CLK / STCLK) x SUMULTIn / 2 / SDPA : n = 0, 1, 2
The figure below shows the settings for generating a 2MHz drive frequency when the PCLKB frequency is 32 MHz. SDPA can be set for each touch interface configuration.

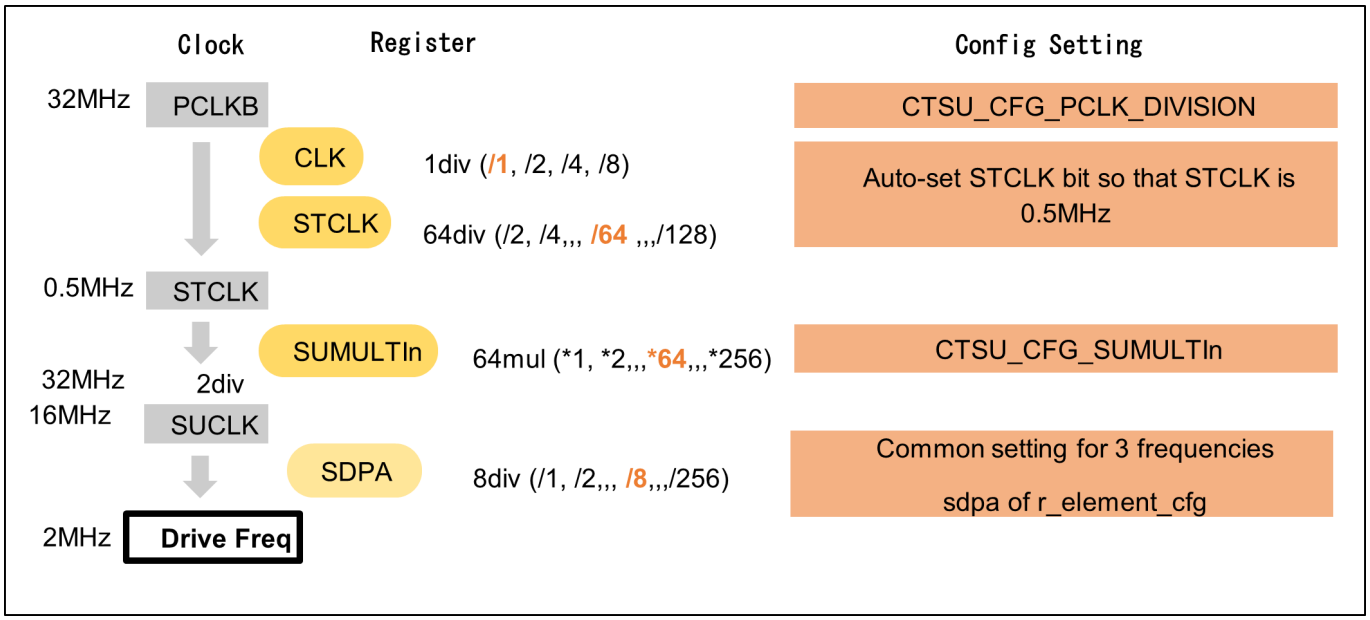



Figure 182: Drive Frequency Settings

### Shield Function(CTSU2)

The CTSU2 peripheral has a built-in function that outputs a shield signal in phase with the drive pulse from the shield terminal and the non-measurement terminal in order to shield against external

influences while suppressing any increase in parasitic capacitance. This function can only be used during self-capacitance measurements.
This module allows the user to set a shield for each touch interface configuration.
For example, for the electrode configuration shown in , the members of ctsu_cfg_t should be set as follows. Other members have been omitted for the example.
.txvsel = CTSU_TXVSEL_INTERNAL_POWER,
.txvsel2 = CTSU_TXVSEL_MODE,
.md = CTSU_MODE_SELF_MULTI_SCAN,
.posel = CTSU_POSEL_SAME_PULSE,
.ctsuchac0 = 0x0F,
.ctsuchtrc0 = 0x08,

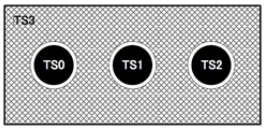



Figure 183: Example of Shield Electrode Structure

### Measurement Error Message

When the CTSU2 peripheral detects an abnormal measurement, it sets the status register bit to 1. In the measurement complete interrupt process, the module reads ICOMP1, ICOMP0, and SENSOVF of the status register and notifies the results in the callback function. The status register is reset after the contents are read. For more details on abnormal measurements, refer to "member event" in the ctsu_callback_args_t callback function argument.

### Moving Average

This function calculates the moving average of the measured results.
Set the number of times the moving average should be calculated in the config settings.

### Diagnosis Function

The CTSU peripheral has a built-in function that diagnoses its own inner circuit. This diagnosis function provides the API for diagnosing the inner circuit.
The diagnostic requirements are different for CTSU1 and CTSU2 providing 5 types of diagnosis for CTSU1 and 9 types for CTSU2.
The diagnosis function is executed by calling the API function. This is executed independently from the other measurements and does not affect them.
To enable the diagnosis function, set CTSU_CFG_DIAG_SUPPORT_ENABLE to 1.
For CTSU1, a 27pF condenser should be connected externally.
For CTSU2, Diagnosis function uses the ADC module.
If an error occurs in the ADC module used for Diagnosis mode, return FSP_ERR_ABORTED as the return value of R_CTSU_DataGet().
If an ADC error is returned, exit the function so as not to measure or close the ADC.
See ADC (r_adc) for ADC module errors.
Please pay particular attention to the following three points.

1. Be sure to measure the ADC module when using the Diagnosis mode function of the CTSU module. Therefore, in order for the user to use it with Dignosis, please close the user's ADC. After closing, please use the Diagnosis mode function of the CTSU module.
2. When creating an application with RTOS, please be careful about the scheduling of the

CTSU module's Diagnosis mode function task and the user's ADC task.
3. If FSP_ERR_ABORTED occurs, please call the user's ADC again when using the user's ADC.

### MEC Function (CTSU2SLa)

The CTSU2SLa peripheral has MEC (Multiple Electrode Connection) function that connects multiple electrodes and measures them as a single electrode. This feature is only available in self capacitance mode.
This is an example when using three electrodes. In normal times, normal measurement is performed, and 3 channels are measured to get each measured value. In power saving, MEC measurement is performed, and one channel is measured by combining three channels to acquire one measured value.
The figure below shows a compare of time of normal measurement and MEC measurement. Since multi channels are measured at the same time, the measurement time is shortened.

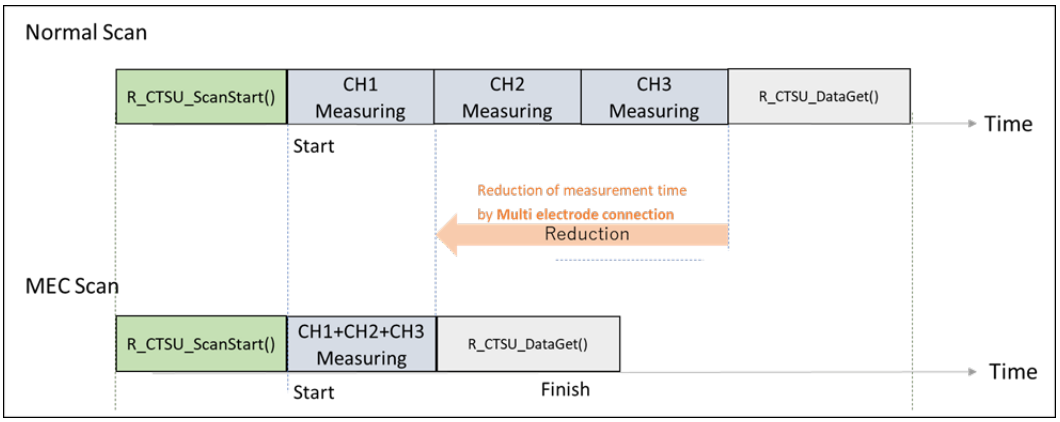


Figure 184: Comparison of time between normal measurement and MEC measurement

To enable the code for the MEC feature, set CTSU_CFG_MULTIPLE_ELECTRODE_CONNECTION_ENABLE to 1.
When using MEC, create a touch interface configuration different from the normal touch interface configuration for the same TS.
The following settings are required for the touch interface configuration for MEC measurement.
To enable MEC for touch interface configurations by setting tsod in ctsu_cfg_t to 1.
Set mec_ts of ctsu_cfg_t to one of the TS numbers to be measured.
If you want to use the shield function at the same time, set the TS number of the shield terminal in mec_shield_ts of ctsu_cfg_t.
In this case, only one TS can be used as a shield terminal. Set num_rx of ctsu_cfg_t to 1.
For example, in the case of the electrode configuration shown below, set the members of ctsu_cfg_t as shown below. Other members are omitted here.
.tsod = 1,
.mec_ts = 0,
.mec_shield_ts = 3,
.num_rx = 1,

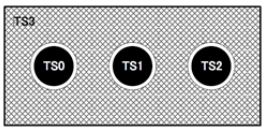



Figure 185: Example of MEC and shield electrode configuration

### Automatic CCO Correction (CTSU2SLa)

CTSU2SL peripheral has an automatic correction that correct the sensor CCO by hardware. Refer to Sensor CCO Correction function for more information on sensor CCO correction.
CTSU2SL peripheral processes the correction calculation. CCO correction data can be calculated without using the correction calculation processing of the software.
The processing time of the main processor is not consumed.

Set CTSU_CFG_AUTO_CORRECTION_ENABLE to 1 to enable this feature.

### Automatic Frequency Correction (CTSU2SLa)

CTSU2SLa peripheral has an automatic frequency correction that correct the calculation by hardware.
CTSU2SLa peripheral processes the correction calculation. Frequency correction data can be calculated without using the correction calculation processing of the software.
The processing time of the main processor is not consumed.

Set CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE to 1 to enable this feature.

### Automatic Judgement (CTSU2SLa)

CTSU2SLa peripheral has an automatic judgement that judges the touch of a button by hardware.
CTSU2SLa peripheral processes the touch judgement of the button. The processing time of the main processor is not consumed.
Measurements and Judgements can be initiated either by software triggers or external events triggered by the Event Link Controller (ELC).
Please use the API function R_CTSU_ScanStart().
This module processes INTCTSUWR and INTCTSURD generated during measurement. Since DTC is used for these processes, DTC is required.

A callback function notifies the application when the processing of INTCTSUFN is completed. Get the judgement result before the next measurement.
Please use the API function R_CTSU_AutoJudgeDataGet().
Select either JMM or VMM as the majority voting method. Below is an example of the ctsu_cfg_t member settings for JMM. For VMM, set "jc" to 0 and "majirimd" to 1.
.tlot = 2, // Non-touch judgement continuous count : 3 times
.thot = 2, // Touch judgement continuous count : 3 times
.jc = 1, // Judgement by two frequency
.ajmmat = 2, // Moving average : 22times
.ajbmat = 7, // Baseline average count : 27+1times
.majirimd = 0, // JMM
.mtucfen = 1, // Mutual-capacitance
.ajfen = 1, // Enable automatic judgement

Set "CTSU_CFG_AUTO_JUDGE_ENABLE = 1" to enable the automatic judgement. In this case, set "CTSU_CFG_AUTO_CORRECTION_ENABLE = 1" to enable the automatic CCO correction function as well.
For VMM, set "CTSU_CFG_AUTO_MULTI_CLOCK_CORRECTION_ENABLE = 1" to enable the automatic frequency correction function as well.

The following (a) to (e) describe the automatic judgement and its setting. In the case of JMM, (a) ~ (e) settings are set for each multi-frequency measurement.

- (a) Measurement mode
  Select self-capacitance or mutual-capacitance with "mtucfen" of ctsu_auto_button_cfg_t. Set the self-capacitance to 0. Set the mutual-capacitance to 1.

- (b) Baseline
  Set the baseline from the measurement result in the non-touch state. After completing the initial offset adjustment with R_CTSU_OffsetTuning (), the baseline is initially set (set BLINI bit) when R_CTSU_ScanStart () is called for the first time.
  After that, when R_CTSU_AutoJudgementDataGet () is called, the baseline initialization is canceled (clear BLINI bit) and the baseline update process is started.
  The baseline is updated every set number of measurements to follow changes in the surrounding environment.
  If "non-touch" state continues for the set number of measurements, the baseline is updated to the average value. When judgement result is "touch", the number of counts is cleared.
  Set the number of measurements (baseline update interval) with "ajbmat" of ctsu_cfg_t.
  Common to all buttons in the touch interface configuration.
  Adjusts the ability to follow changes in the surrounding environment.

- (c) Threshold
  judgement is made using a threshold with an arbitrary offset from the baseline. The threshold is set by adding hysteresis.
  Chattering is prevented by giving hysteresis to the transition from "touch" to "non-touch". Increasing the hysteresis value is more effective in preventing chattering, but be aware that it will be more difficult to transition from "touch" to "non-touch".
  Set the threshold and hysteresis for each button with threshold and hysteresis of ctsu_auto_button_cfg_t.
  This module calculates the upper threshold and the lower threshold from these and sets them in the CTSUAJTHR register.
  The figure below shows the self-capacitance judgement. Since the electrode capacitance of the self-capacitance button increases when touched, it is judged "touch" when the upper threshold is exceeded.

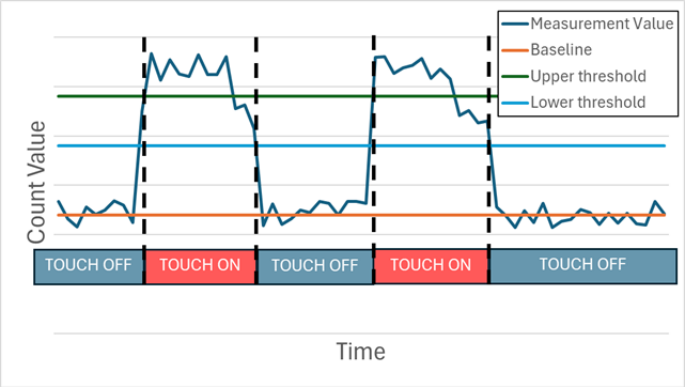



Figure 186: Self-capacitance judgement

The figure below shows the mutual-capacitance judgement. Since the mutual capacitance button reduces the capacitance between electrodes when touched, it is judged as "touch" when the lower threshold is exceeded.

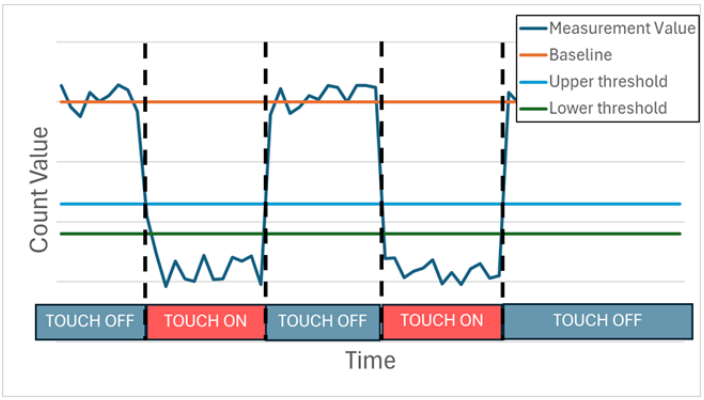



Figure 187: Mutual-capacitance judgement

- (d) The number of consecutive "non-touch" and "touch" detections
  This is a filter function to judge "touch" or "non-touch" when "touch" or "non-touch" state continues for a certain number of times.
  Set the number of times with "tlot" and "thot" of ctsu_cfg_t. Common to all buttons in the touch interface configuration.
  Increasing the number of consecutive times will be more effective against chattering, but be aware that the reaction speed will decrease.

- (e) Moving average
  With the automatic judgement function, Set the number of moving averages with "ajmmat" of ctsu_cfg_t.Common to all buttons in the touch interface configuration.
  The figure below shows the button judgement operation described above.

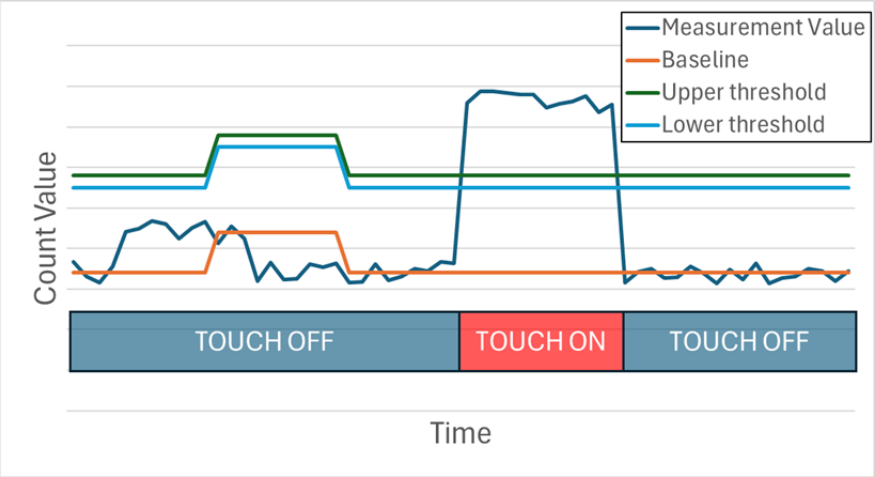



Figure 188: Button judgement

### Measurement Mode

This module supports all three modes offered by the CTSU2 peripheral: self-capacitance, mutual-capacitance, and current measurement modes. The temperature correction mode is also offered as a mode for updating the correction coefficient.

### Self-capacitance Mode

The self-capacitance mode is used to measure the capacitance of each terminal (TS).
The CTSU2 peripheral measures the terminals in ascending order according to the TS numbers, then stores the data. For example, even if you want to use TS5, TS8, TS2, TS3 and TS6 in your application in that order, they will still be measured and stored in the order of TS2, TS3, TS5, TS6, and TS8. Therefore, you will need to reference buffer indexes [2], [4], [0], [1], and [3].
VMM returns one measurement result for each element.
JMM returns three measurement results for each element.

[CTSU1]
In default settings, the measurement period for each TS is wait-time plus approximately 526us.

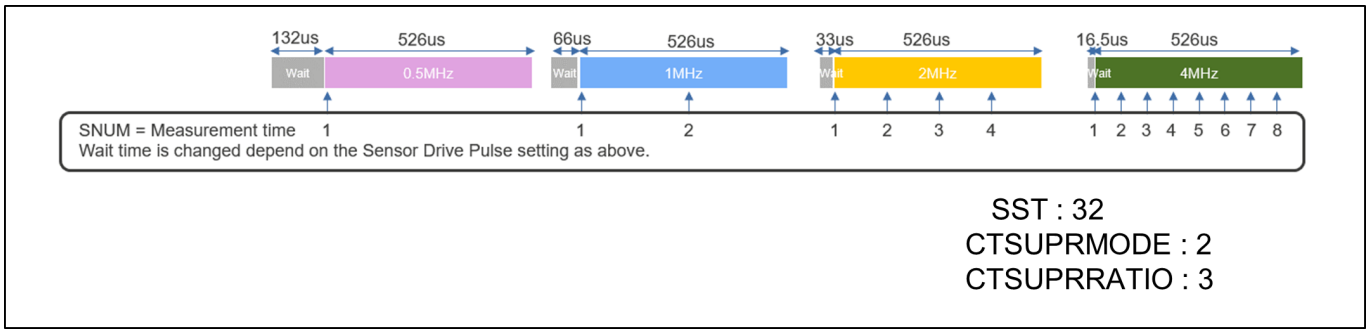



Figure 189: Self-capacitance Measurement Period (CTSU1)

[CTSU2]
In default settings, the measurement period for each TS is approximately 576us.

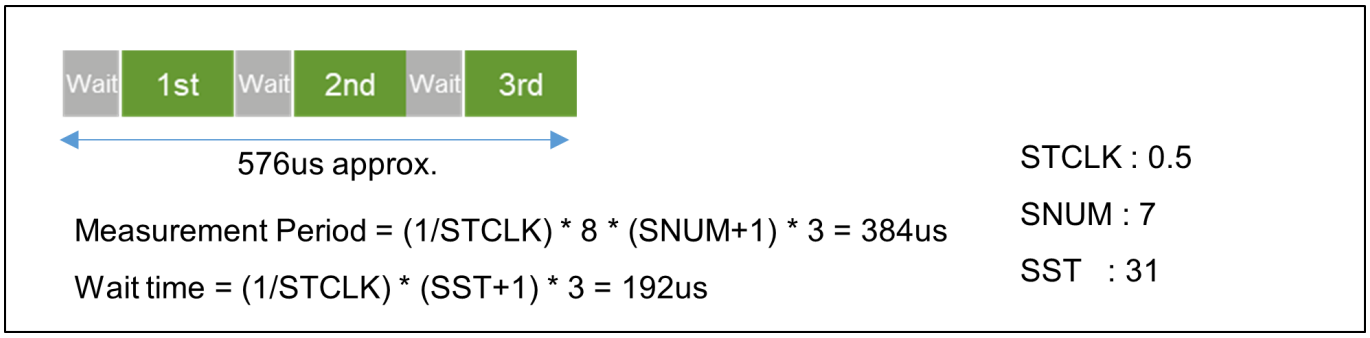



Figure 190: Self-capacitance Measurement Period (CTSU2)

### Mutual-Capacitance Mode

The mutual-capacitance mode is used to measure the capacitance generated between the receive TS (Rx) and transmit TS (Tx), and therefore requires at least two terminals.
The CTSU2 peripheral measures all specified combinations of Rx and Tx. For example, when Rx is TS1 and TS3, and Tx is TS2, TS7 and TS4, the combinations are measured in the following order and the data is stored.
TS3-TS2, TS3-TS4, TS3-TS7, TS10-TS2, TS10-TS4, TS10-TS7
To measure the mutual-capacitance generated between electrodes, the CTSU2 peripheral performs the measurement process on the same electrode twice.
The mutual-capacitance is obtained by inverting the phase relationship of the pulse output and switched capacitor in the primary and secondary measurements, and calculating the difference between the two measurements. This module does not calculate the difference, but outputs both primary and secondary measured result.
VMM returns two measurement results(one primary and one secondary) for each element.
JMM returns six measurement results(three primary and three secondary) for each element.
[CTSU1]
In default settings, the measurement period for each TS is twice of wait-time plus approximately 526us.
[CTSU2]
In default settings, the measurement period for each TS is approximately 1152us.

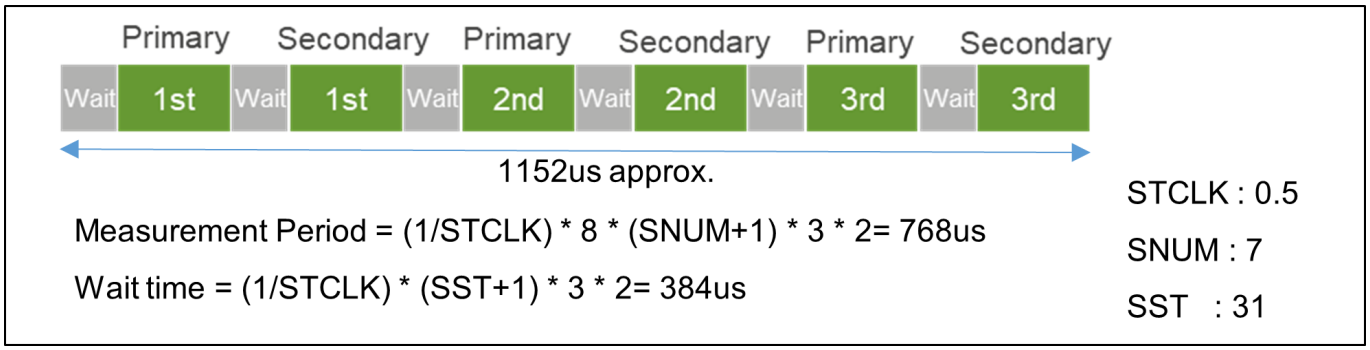



Figure 191: Mutual-capacitance Measurement Period

### Mutual-capacitance parallel scan mode(CTSU2)

This mode provides fast measurement time by parallel scanning the RX lines with a CFC circuit. Operation is otherwise identical to normal CTSU mutual scanning.

- Scan Order
  - The hardware scans all RX pins simultaneously for each TX pin.
  - For example, if sensors TS10, TS11, and TS03 are specified as RX sensors, and sensors TS02, TS07, and TS04 are specified as TX sensors, the hardware will scan them in the following sensor-pair order:
    TS02-(TS03, TS10, TS11), TS04-(TS03, TS10, TS11), TS07-(TS03, TS10, TS11)
- Element
  - An element refers to the index of a sensor-pair within the scan order. Using the previous example, TS07-TS10 is element 7.
- Scan Time
  - Because the RX lines are scanned in parallel, CFC mutual-capacitance scan is the same amount of times faster than a basic mutual matrix scan as the number of RX lines. In other words, on a matrix with N receive lines, CFC mutual scanning is N times faster than basic mutual scanning. Set CTSU_MODE_MUTUAL_CFC_SCAN to "md" of ctsu_cfg_t.
    Also, add the number of matrix used for this measurement to CTSU_CFG_NUM_MUTUAL_ELEMENTS. In addition, set the number of CTSU_CFG_NUM_CFC and CTSU_CFG_NUM_CFC_TX.
    For details, refer to the configuration and sample application output by QE for Capacitive Touch.

**Current Measurement Mode(CTSU2)**

The current measurement mode is used to measure the minute current input to the TS terminal. The order of measurement and data storage is the same as that of the self-capacitance mode. As this does not involve the switched capacitor operation, the measurement is only performed once. The measurement period for one TS under default settings is approximately 256us. The current measurement mode requires a longer stable wait time than the other modes, so the SST is set to 63.

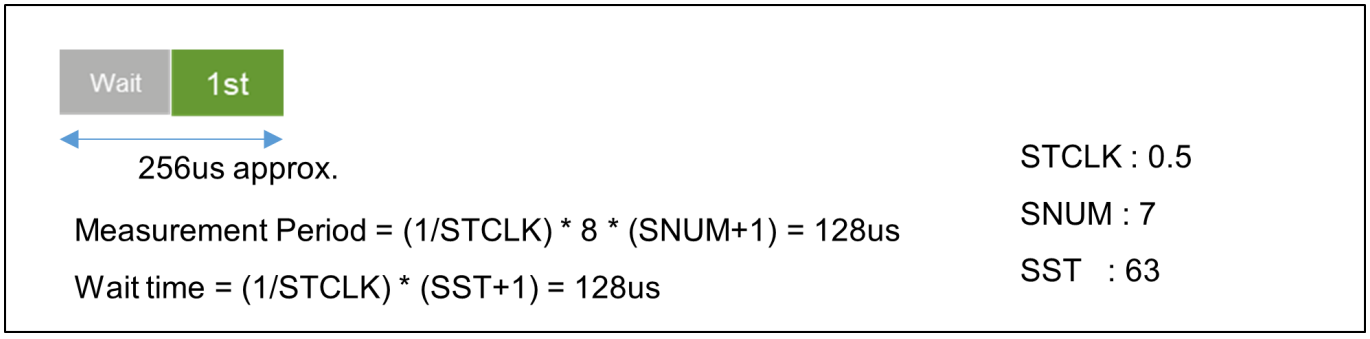



Figure 192: Current Measurement Period

**Temperature Correction Mode(CTSU2)**

The temperature correction mode is used to periodically update the correction coefficient using an external resistor connected to a TS terminal. This involves three processes as described below. Also refer to the timing chart in Figure of Temperature Correction Measurement Timing Chart.

1. Measure the correction circuit. One set comprises twelve measurements.
2. Measure the current when TSCAP voltage is applied to the external resistor to create a

correction coefficient based on an external resistor that does not depend on temperature. Execute the next measurement after the previous measurement set is completed (as described in step 1).
3. Flow offset current to the external resistor and measure the voltage with the ADC. This will adjust the RTRIM register and handle the temperature drift of the internal reference resistor. In the config settings, set the number of times step 2 should be executed before carrying out this measurement.

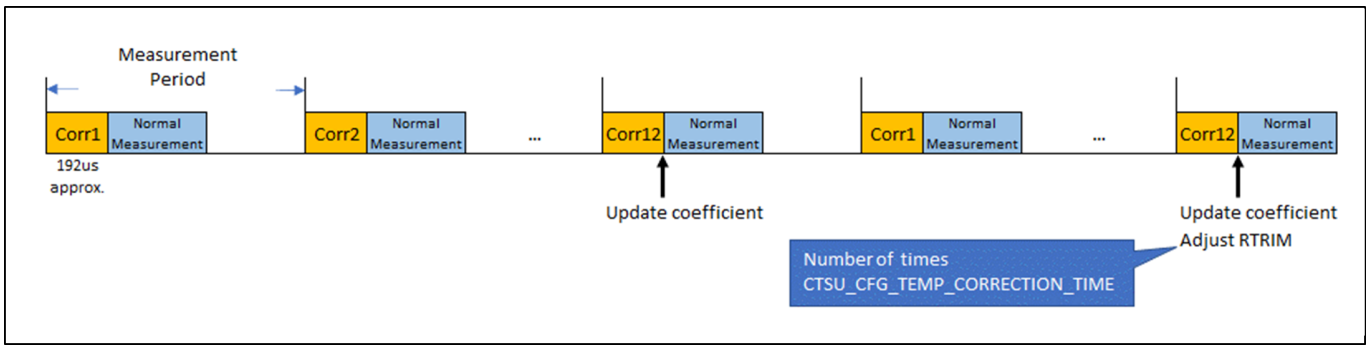



Figure 193: Temperature Correction Measurement Timing Chart

Temperature correction uses the ADC module.
If an error occurs in the ADC module used for temperature correction, return FSP_ERR_ABORTED as the return value of R_CTSU_DataGet().
If an ADC error is returned, exit the function so as not to measure or close the ADC.
See ADC (r_adc) for ADC module errors.
Please pay articular attention to the following three points.

1. When using the temperature correction of the CTSU module, be sure to measure the ADC module. Therefore, please close the user's ADC for use in temperature correction. After closing, please use temperature correction of CTSU module.
2. When creating an RTOS, please be careful about the scheduling of the CTSU module's temperature correction task and the user's ADC task when creating an application.
3. If FSP_ERR_ABORTED occurs, please call the user's ADC again when using the user's ADC.

**Diagnosis Mode**

The diagnosis mode is a mode in which various internal measurement values are scanned by using this diagnosis function R_CTSU_Diagnosis().

**Measurement Timing**

Measurements are initiated by a software trigger or an external event which is triggered by the Event Link Controller (ELC).
The most common method is using a timer to carry out periodic measurements. Make sure to set the timer interval to allow the measurement and internal value update processes to complete before the next measurement period. The measurement period differs according to touch interface configuration and measurement mode.
The execution timing of software triggers and external triggers differ slightly.
Since a software trigger sets the start flag after setting the touch interface configuration with R_CTSU_ScanStart(), there is a slight delay after the timer event occurrence. However, as the delay is much smaller than the measurement period, a software trigger is recommended for most instances as it is easy to set.

An external trigger is recommended for applications in which this slight delay is not acceptable or that require low-power consumption operations. When using an external trigger with multiple touch interface configurations, use R_CTSU_ScanStart() to set another touch interface configuration after one measurement is completed.

### TrustZone Support

In r_ctsu and rm_touch module, Non-Secure Callable Guard Functions are only generated from QE for Capacitive Touch. QE can be used for tuning in secure or flat project, but not in non-secure project. If you want to use in non-secure project, copy the output file from secure or flat project. Refer to QE Help for more information.

### Data flow

The flow of storing data in RAM is as follows.

(CTSU1)

1. Read registers and stored in RAM as raw data.
2. ICO correction calculation of raw data and stored in RAM as correction data.
3. The correction data is calculated by moving average and stored in RAM as measurement results.

(CTSU2 VMM)

1. Reads a register and stores raw data measured at three different frequencies in RAM.
2. Calculate CCO correction from three raw data and stored three CCO correction values in RAM.
3. Calculate three frequency correction values from three CCO correction values.
4. Three frequency correction data are calculated by majority decision and moving average, and stored in RAM as measurement results.

(CTSU2 JMM)

1. Reads a register and stores raw data measured at three different frequencies in RAM.
2. Calculate CCO correction from three raw data and stored three CCO correction values in RAM.
3. Three CCO correction data is calculated by moving average and stored three measurement results in RAM.

### Add user's filter

There are two ways to add the user's filter.

1. Instead of filter calculation of R_CTSU_DataGet(), perform user filter calculation and use R_CTSU_DataInsert() to input user filter calculation result.
   R_CTSU_DataInsert()
   In VMM, the second argument p_insert_data stores one data for each element.
   In JMM, the second argument p_insert_data stores three data for each element.

2. Using the correction data obtained by R_CTSU_SpecificDataGet(), instead of majority

decision calculation and filter calculation of R_CTSU_DataGet(), perform user majority decision calculation & filter calculation and use R_CTSU_DataInsert() to input user majority decision calculation & filter calculation result.

Please check example.
User's filter additional Example

### Limitations

Developers should be aware of the following limitations when using CTSU with FSP :

- CTSU doesn't support Diagnosis for CTSU2SLa.

## Examples

### Basic Example

This is a basic example of minimal use of the CTSU in an application.

```
volatile bool g_scan_flag = false;
void ctsu_callback (ctsu_callback_args_t * p_args)
{
 if (CTSU_EVENT_SCAN_COMPLETE == p_args->event)
    {
       g_scan_flag = true;
    }
}
void ctsu_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
```

```
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
    }
}
```

### Multi-configuration Example

This is a optional example of using both Self-capacitance and Mutual-capacitance configurations in the same project.

```
void ctsu_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS * 2)];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    assert(FSP_SUCCESS == err);
    err = R_CTSU_Open(&g_ctsu_ctrl_mutual, &g_ctsu_cfg_mutual);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
 R_CTSU_ScanStart(&g_ctsu_ctrl_mutual);
 while (!g_scan_flag)
      {
```

```
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
       err = R_CTSU_DataGet(&g_ctsu_ctrl_mutual, data);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
    }
}
```

### Offset Adjustment Example

This is an example of offset adjustment using R_CTSU_OffsetTuning().
After completing R_CTSU_Open (), perform initial offset adjustment.
Offset adjustment is performed again when the parasitic capacitance changes significantly due to changes in the surrounding environment and the count value becomes an abnormal value.

```
void ctsu_offsettuning_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    assert(FSP_SUCCESS == err);
 /* Initial offset tuning */
 do
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
```

```
     {
 /* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_OffsetTuning(&g_ctsu_ctrl);
   } while (FSP_SUCCESS != err);
 while (true)
   {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
     {
 /* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
     assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
     {
 /* Re-offset tuning is performed when the parasitic capacitance changes
significantly due   */
 /* to changes in the surrounding environment and the count value becomes an abnormal
value. */
 /*
       */
 /* if (abnormal value detection
conditions)                                                */
 /* {
       */
 /* Re-offset tuning */
 do
     {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
     {
```

```
 /* Wait for scan end callback */
      }
         g_scan_flag = false;
         err = R_CTSU_OffsetTuning(&g_ctsu_ctrl);
      } while (FSP_SUCCESS != err);
 /* }
        */
      }
    }
}
```

### Diagnosis function Example

This is a Diagnosis function example of using the configuration in the basic example.

```
void ctsu_diag_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t dummy;
 R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    assert(FSP_SUCCESS == err);
 R_CTSU_Open(&g_ctsu_ctrl_diagnosis, &g_ctsu_cfg_diagnosis);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
      assert(FSP_SUCCESS == err);
 R_CTSU_ScanStart(&g_ctsu_ctrl_diagnosis);
```

```
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl_diagnosis, &dummy);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
            err = R_CTSU_Diagnosis(&g_ctsu_ctrl_diagnosis);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
 break;
      }
      }
    }
}
```

### User's filter additional Example

This is a user's filter additiional example of using the configuration in the basic example.
To perform user's filter calculation, change the num_moving_average of the element in the target ctsu_cfg_t to 1.

Perform user filter calculation and use R_CTSU_DataInsert() to input user filter calculation result.

```
void ctsu_user_filter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t filter_data[CTSU_CFG_NUM_SELF_ELEMENTS];
 /* If you want to make a touch judgment, call RM_TOUCH_Open()instead of the
following. */
```

```
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* If you want to make a touch judgment, call RM_TOUCH_ScanStart()instead of the
following. */
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
 /* User original function. */
            ctsu_user_filter(data, filter_data);
            err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
      assert(FSP_SUCCESS == err);
 /* Call RM_TOUCH_DataGet() to make a touch decision. */
      }
    }
}
```

Using the correction data obtained by R_CTSU_SpecificDataGet(). Perform user majority decision calculation & filter calculation and use R_CTSU_DataInsert() to input user majority decision calculation & filter calculation result.

```
void ctsu_user_majority_decition_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t corr_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI];
```

```
    uint16_t filter_data[CTSU_CFG_NUM_SELF_ELEMENTS];
 /* If you want to make a touch judgment, call RM_TOUCH_Open()instead of the
following. */
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* If you want to make a touch judgment, call RM_TOUCH_ScanStart()instead of the
following. */
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
            err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, corr_data,
CTSU_SPECIFIC_CORRECTION_DATA);
      assert(FSP_SUCCESS == err);
 /* User original function */
            ctsu_user_filter(corr_data, filter_data);
            err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
      assert(FSP_SUCCESS == err);
 /* Call RM_TOUCH_DataGet() to make a touch decision. */
      }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | ctsu_ctsuwr_t |

| | |
|---|---|
| struct | ctsu_self_buf_t |
| struct | ctsu_mutual_buf_t |
| struct | ctsu_correction_info_t |
| struct | ctsu_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ctsu_state_t |
| enum | ctsu_tuning_t |
| enum | ctsu_correction_status_t |
| enum | ctsu_range_t |

## Data Structure Documentation

### ◆ ctsu_ctsuwr_t

| struct ctsu_ctsuwr_t | | |
|---|---|---|
| CTSUWR write register value | | |
| Data Fields | | |
| uint16_t | ctsussc | Copy from (ssdiv << 8) by Open API. |
| uint16_t | ctsuso0 | Copy from ((snum << 10) \| so) by Open API. |
| uint16_t | ctsuso1 | Copy from (sdpa << 8) by Open API. ICOG and RICOA is set recommend value. |

### ◆ ctsu_self_buf_t

| struct ctsu_self_buf_t | | |
|---|---|---|
| Scan buffer data formats (Self) | | |
| Data Fields | | |
| uint16_t | sen | Sensor counter data. |
| uint16_t | ref | Reference counter data (Not used) |

### ◆ ctsu_mutual_buf_t

| struct ctsu_mutual_buf_t |
|---|
| Scan buffer data formats (Mutual) |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | pri_sen | Primary sensor data. |
| uint16_t | pri_ref | Primary reference data (Not used) |
| uint16_t | snd_sen | Secondary sensor data. |
| uint16_t | snd_ref | Secondary reference data (Not used) |

### ◆ ctsu_correction_info_t

| struct ctsu_correction_info_t | | |
|---|---|---|
| Correction information | | |
| Data Fields | | |
| ctsu_correction_status_t | status | Correction status. |
| ctsu_ctsuwr_t | ctsuwr | Correction scan parameter. |
| volatile ctsu_self_buf_t | scanbuf | Correction scan buffer. |
| uint8_t | calculation_error | Overflow or underflow in correction calclation. |
| uint16_t | first_val | 1st correction value |
| uint16_t | second_val | 2nd correction value |
| uint32_t | first_coefficient | 1st correction coefficient |
| uint32_t | second_coefficient | 2nd correction coefficient |
| uint32_t | ctsu_clock | CTSU clock [MHz]. |

### ◆ ctsu_instance_ctrl_t

| struct ctsu_instance_ctrl_t | |
|---|---|
| CTSU private control block. DO NOT MODIFY. Initialization occurs when R_CTSU_Open() is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Whether or not driver is open. |
| | |
| volatile ctsu_state_t | state |
| | CTSU run state. |
| | |
| ctsu_cap_t | cap |
| | CTSU Scan Start Trigger Select. |
| | |

| | |
|---|---|
| ctsu_md_t | md |
| | CTSU Measurement Mode Select(copy to cfg) |
| | |
| ctsu_tuning_t | tuning |
| | CTSU Initial offset tuning status. |
| | |
| uint16_t | num_elements |
| | Number of elements to scan. |
| | |
| uint16_t | wr_index |
| | Word index into ctsuwr register array. |
| | |
| uint16_t | rd_index |
| | Word index into scan data buffer. |
| | |
| uint8_t * | p_element_complete_flag |
| | Pointer to complete flag of each element.<br>g_ctsu_element_complete_flag[] is set by Open API. |
| | |
| int32_t * | p_tuning_diff |
| | Pointer to difference from base value of each element.<br>g_ctsu_tuning_diff[] is set by Open API. |
| | |
| uint16_t | average |
| | CTSU Moving average counter. |
| | |
| uint16_t | num_moving_average |
| | Copy from config by Open API. |

| | |
|---|---|
| | |
| uint8_t | ctsucr1 |
| | Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW, and PON is set by HAL driver. |
| | |
| ctsu_ctsuwr_t * | p_ctsuwr |
| | CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API. |
| | |
| ctsu_self_buf_t * | p_self_raw |
| | Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API. |
| | |
| uint16_t * | p_self_corr |
| | Pointer to Self correction data. g_ctsu_self_corr[] is set by Open API. |
| | |
| uint16_t * | p_self_mfc |
| | Pointer to Self multi frequency correction data. g_ctsu_self_mfc[] is set by Open API. |
| | |
| ctsu_data_t * | p_self_data |
| | Pointer to Self moving average data. g_ctsu_self_data[] is set by Open API. |
| | |
| ctsu_mutual_buf_t * | p_mutual_raw |
| | Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open API. |
| | |
| uint16_t * | p_mutual_pri_corr |
| | Pointer to Mutual primary correction data. g_ctsu_self_corr[] is set by Open API. |
| | |
| uint16_t * | p_mutual_snd_corr |

| | |
|---|---|
| | Pointer to Mutual secondary correction data. g_ctsu_self_corr[] is set by Open API. |
| | |
| uint16_t * | p_mutual_pri_mfc |
| | Pointer to Mutual primary multi frequency correction data. g_ctsu_pri_mutual_mfc[] is set by Open API. |
| | |
| uint16_t * | p_mutual_snd_mfc |
| | Pointer to Mutual primary multi frequency correction data. g_ctsu_pri_mutual_mfc[] is set by Open API. |
| | |
| ctsu_data_t * | p_mutual_pri_data |
| | Pointer to Mutual primary moving average data. g_ctsu_mutual_pri_data[] is set by Open API. |
| | |
| ctsu_data_t * | p_mutual_snd_data |
| | Pointer to Mutual secondary moving average data. g_ctsu_mutual_snd_data[] is set by Open API. |
| | |
| ctsu_correction_info_t * | p_correction_info |
| | Pointer to correction info. |
| | |
| ctsu_txvsel_t | txvsel |
| | CTSU Transmission Power Supply Select. |
| | |
| ctsu_txvsel2_t | txvsel2 |
| | CTSU Transmission Power Supply Select 2 (CTSU2 Only) |
| | |
| uint8_t | ctsuchac0 |
| | TS00-TS07 enable mask. |

| | |
|---|---|
| | |
| uint8_t | ctsuchac1 |
| | TS08-TS15 enable mask. |
| | |
| uint8_t | ctsuchac2 |
| | TS16-TS23 enable mask. |
| | |
| uint8_t | ctsuchac3 |
| | TS24-TS31 enable mask. |
| | |
| uint8_t | ctsuchac4 |
| | TS32-TS39 enable mask. |
| | |
| uint8_t | ctsuchtrc0 |
| | TS00-TS07 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc1 |
| | TS08-TS15 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc2 |
| | TS16-TS23 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc3 |
| | TS24-TS31 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc4 |
| | TS32-TS39 mutual-tx mask. |
| | |

| | |
|---|---|
| uint16_t | self_elem_index |
| | self element index number for Current instance. |
| | |
| uint16_t | mutual_elem_index |
| | mutual element index number for Current instance. |
| | |
| uint16_t | ctsu_elem_index |
| | CTSU element index number for Current instance. |
| | |
| ctsu_cfg_t const * | p_ctsu_cfg |
| | Pointer to initial configurations. |
| | |
| IRQn_Type | write_irq |
| | Copy from config by Open API. CTSU_CTSUWR interrupt vector. |
| | |
| IRQn_Type | read_irq |
| | Copy from config by Open API. CTSU_CTSURD interrupt vector. |
| | |
| IRQn_Type | end_irq |
| | Copy from config by Open API. CTSU_CTSUFN interrupt vector. |
| | |
| void(* | p_callback )(ctsu_callback_args_t *) |
| | Callback provided when a CTSUFN occurs. |
| | |
| uint8_t | interrupt_reverse_flag |
| | Flag in which read interrupt and end interrupt are reversed. |
| | |
| ctsu_event_t | error_status |

| | |
|---|---|
| | error status variable to send to QE for serial tuning. |
| | |
| ctsu_callback_args_t * | p_callback_memory |
| | Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| bool | serial_tuning_enable |
| | Flag of serial tuning status. |
| | |
| uint16_t | serial_tuning_mutual_cnt |
| | Word index into ctsuwr register array. |
| | |
| uint16_t | tuning_self_target_value |
| | Target self value for initial offset tuning. |
| | |
| uint16_t | tuning_mutual_target_value |
| | Target mutual value for initial offset tuning. |
| | |
| uint8_t | tsod |
| | Copy from tsod by Open API. |
| | |
| uint8_t | mec_ts |
| | Copy from mec_ts by Open API. |
| | |
| uint8_t | mec_shield_ts |
| | |

| | |
|---|---|
| | Copy from mec_shield_ts by Open API. |
| | |

## Enumeration Type Documentation

### ◆ ctsu_state_t

| enum ctsu_state_t | |
|---|---|
| CTSU run state | |
| Enumerator | |
| CTSU_STATE_INIT | Not open. |
| CTSU_STATE_IDLE | Opened. |
| CTSU_STATE_SCANNING | Scanning now. |
| CTSU_STATE_SCANNED | Scan end. |

### ◆ ctsu_tuning_t

| enum ctsu_tuning_t | |
|---|---|
| CTSU Initial offset tuning status | |
| Enumerator | |
| CTSU_TUNING_INCOMPLETE | Initial offset tuning incomplete. |
| CTSU_TUNING_COMPLETE | Initial offset tuning complete. |

### ◆ ctsu_correction_status_t

| enum ctsu_correction_status_t | |
|---|---|
| CTSU Correction status | |
| Enumerator | |
| CTSU_CORRECTION_INIT | Correction initial status. |
| CTSU_CORRECTION_RUN | Correction scan running. |
| CTSU_CORRECTION_COMPLETE | Correction complete. |
| CTSU_CORRECTION_ERROR | Correction error. |

### ◆ ctsu_range_t

| enum ctsu_range_t | |
|---|---|
| CTSU range definition | |
| Enumerator | |
| CTSU_RANGE_20UA | 20uA mode |
| CTSU_RANGE_40UA | 40uA mode |
| CTSU_RANGE_80UA | 80uA mode |
| CTSU_RANGE_160UA | 160uA mode |
| CTSU_RANGE_NUM | number of range |

## Function Documentation

### ◆ R_CTSU_Open()

fsp_err_t R_CTSU_Open ( ctsu_ctrl_t *const *p_ctrl*, ctsu_cfg_t const *const *p_cfg* )

Opens and configures the CTSU driver module. Implements ctsu_api_t::open.

Example:

```
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

- *In the first Open, measurement for correction works, and it takes several tens of milliseconds.*
- *When the touch interface configuration is diagnosis mode, execute the R_CTSU_Open() of the other touch interface configuration first.*

### ◆ R_CTSU_ScanStart()

fsp_err_t R_CTSU_ScanStart ( ctsu_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     assert(FSP_SUCCESS == err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance or other. |
| FSP_ERR_CTSU_NOT_GET_DATA | The previous data has not been retrieved by DataGet. |

### ◆ R_CTSU_DataGet()

fsp_err_t R_CTSU_DataGet ( ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_data* )

This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     assert(FSP_SUCCESS == err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |
| FSP_ERR_CTSU_DIAG_NOT_YET | Diagnosis of data collected no yet. |
| FSP_ERR_INVALID_MODE | The mode of automatic judgement on is invalid. |
| FSP_ERR_ABORTED | Operate error of Diagnosis ADC data collection ,since ADC use other |

### ◆ R_CTSU_AutoJudgementDataGet()

fsp_err_t R_CTSU_AutoJudgementDataGet ( ctsu_ctrl_t *const *p_ctrl*, uint64_t * *p_button_status* )

This function gets the result of automatic judgement button. Call after the scan is completed. The result is 64-bit bitmaps and is stored in order of TS number for specified ctsu control. After the initial judgement, the baseline initialization bit is set and the automatic judgement threshold is set. This function is called only for automatic judgement. Implements ctsu_api_t::autoJudgementDataGet.

Example:

```
/* Main loop */
while (true)
   {
R_CTSU_ScanStart(&g_ctsu_ctrl);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_AutoJudgementDataGet(&g_ctsu_ctrl, &button_status);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_INVALID_MODE | The mode of automatic judgement off is invalid. |

### ◆ R_CTSU_OffsetTuning()

fsp_err_t R_CTSU_OffsetTuning ( ctsu_ctrl_t *const *p_ctrl*)

This function tunes the offset register(SO). Call after the measurement is completed. If the return value is FSP_ERR_CTSU_INCOMPLETE_TUNING, tuning is not complete. Execute the measurement and this function call routine until the return value becomes FSP_SUCCESS. It is recommended to run this routine after R_CTSU_Open(). It can be recalled and tuned again. When the automatic judgement is enabled, after the offset tuning is completed,the baseline initialization bit flag is set. Implements ctsu_api_t::offsetTuning.

Example:

```
/* Initial offset tuning */
do
   {
R_CTSU_ScanStart(&g_ctsu_ctrl);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_OffsetTuning(&g_ctsu_ctrl);
   } while (FSP_SUCCESS != err);
while (true)
   {
R_CTSU_ScanStart(&g_ctsu_ctrl);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
     assert(FSP_SUCCESS == err);
if (FSP_SUCCESS == err)
     {
/* Re-offset tuning is performed when the parasitic capacitance changes
significantly due   */
/* to changes in the surrounding environment and the count value becomes an abnormal
```

```
value. */
 /*
        */
 /* if (abnormal value detection
conditions)                                                */
 /* {
        */
 /* Re-offset tuning */
 do
     {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
     {
 /* Wait for scan end callback */
     }
        g_scan_flag = false;
        err = R_CTSU_OffsetTuning(&g_ctsu_ctrl);
     } while (FSP_SUCCESS != err);
 /* }
        */
     }
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ R_CTSU_ScanStop()

fsp_err_t R_CTSU_ScanStop ( ctsu_ctrl_t *const *p_ctrl*)

This function scan stops the sensor as scanning by the CTSU. Implements ctsu_api_t::scanStop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully scan stop. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CTSU_CallbackSet()

fsp_err_t R_CTSU_CallbackSet ( ctsu_ctrl_t *const *p_api_ctrl*, void(*)(ctsu_callback_args_t *) *p_callback*, void *const *p_context*, ctsu_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements ctsu_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_CTSU_Close()

fsp_err_t R_CTSU_Close ( ctsu_ctrl_t *const *p_ctrl*)

Disables specified CTSU control block. Implements ctsu_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CTSU_SpecificDataGet()

fsp_err_t R_CTSU_SpecificDataGet ( ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_specific_data*, ctsu_specific_data_type_t *specific_data_type* )

This function gets the sensor specific data values as scanned by the CTSU. Call this function after calling the R_CTSU_DataGet() function.

By setting the third argument to CTSU_SPECIFIC_RAW_DATA, RAW data can be output from the second argument.

By setting the third argument to CTSU_SPECIFIC_CCO_CORRECTION_DATA, the cco corrected data can be output from the second argument.

By setting the third argument to CTSU_SPECIFIC_CORRECTION_DATA, the frequency corrected data can be output from the second argument.

By setting the third argument to CTSU_SPECIFIC_SELECTED_FREQ, Get bitmap of the frequency values used in majority decision from the second argument.(CTSU2 Only) The bitmap is shown as follows.

| 2bit | 1bit | 0bit |
|---|---|---|
| 3rd frequency value | 2nd frequency value | 1st frequency value |

Implements ctsu_api_t::specificDataGet.

Example:

```
            err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, corr_data,
CTSU_SPECIFIC_CORRECTION_DATA);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |
| FSP_ERR_NOT_ENABLED | CTSU_SPECIFIC_SELECTED_FREQ is not enabled in CTSU1.(CTSU2 Only) |

### ◆ R_CTSU_DataInsert()

fsp_err_t R_CTSU_DataInsert ( ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_insert_data* )

This function inserts the value of the second argument as the measurement result value. Call this function after calling the R_CTSU_DataInsert() function. Implements ctsu_api_t::dataInsert.

Example:

```
err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ R_CTSU_Diagnosis()

fsp_err_t R_CTSU_Diagnosis ( ctsu_ctrl_t *const *p_ctrl*)

Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis.

Example:

```
            err = R_CTSU_Diagnosis(&g_ctsu_ctrl_diagnosis);
        assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CTSU successfully configured. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_NOT_GET_DATA | The previous data has not been retrieved by DataGet. |
| FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE | Diagnosis of LDO over voltage failed. |
| FSP_ERR_CTSU_DIAG_CCO_HIGH | Diagnosis of CCO into 19.2uA failed. |
| FSP_ERR_CTSU_DIAG_CCO_LOW | Diagnosis of CCO into 2.4uA failed. |
| FSP_ERR_CTSU_DIAG_SSCG | Diagnosis of SSCG frequency failed. |
| FSP_ERR_CTSU_DIAG_DAC | Diagnosis of non-touch count value failed. |
| FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE | Diagnosis of LDO output voltage failed. |
| FSP_ERR_CTSU_DIAG_OVER_VOLTAGE | Diagnosis of over voltage detection circuit failed. |
| FSP_ERR_CTSU_DIAG_OVER_CURRENT | Diagnosis of over current detection circuit failed. |
| FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE | Diagnosis of LDO internal resistance value failed. |
| FSP_ERR_CTSU_DIAG_CURRENT_SOURCE | Diagnosis of LDO internal resistance value failed. |
| FSP_ERR_CTSU_DIAG_SENSCLK_GAIN | Diagnosis of SENSCLK frequency gain failed. |
| FSP_ERR_CTSU_DIAG_SUCLK_GAIN | Diagnosis of SUCLK frequency gain failed. |
| FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY | Diagnosis of SUCLK clock recovery function failed. |
| FSP_ERR_CTSU_DIAG_CFC_GAIN | Diagnosis of CFC oscillator gain failed. |

## 5.2.5.2 Touch (rm_touch)

Modules » CapTouch

## Functions

| | |
|---|---|
| fsp_err_t | RM_TOUCH_Open (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)<br>Opens and configures the TOUCH Middle module. Implements touch_api_t::open. More... |
| fsp_err_t | RM_TOUCH_ScanStart (touch_ctrl_t *const p_ctrl)<br>This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart. More... |
| fsp_err_t | RM_TOUCH_DataGet (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)<br>Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet. More... |
| fsp_err_t | RM_TOUCH_PadDataGet (touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch)<br>This function gets the current position of pad is being pressed. Implements touch_api_t::padDataGet , g_touch_on_ctsu. More... |
| fsp_err_t | RM_TOUCH_ScanStop (touch_ctrl_t *const p_ctrl)<br>Scan stop specified TOUCH control block. Implements touch_api_t::scanStop. More... |
| fsp_err_t | RM_TOUCH_CallbackSet (touch_ctrl_t *const p_api_ctrl, void(*p_callback)(touch_callback_args_t *), void *const p_context, touch_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_TOUCH_Close (touch_ctrl_t *const p_ctrl)<br>Disables specified TOUCH control block. Implements touch_api_t::close. More... |

| | |
|---|---|
| fsp_err_t | RM_TOUCH_SensitivityRatioGet (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)<br>Get the touch sensitivity ratio. Implements touch_api_t::sensitivityRatioGet. More... |
| fsp_err_t | RM_TOUCH_ThresholdAdjust (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)<br>Adjust the touch judgment threshold. Implements touch_api_t::thresholdAdjust. More... |
| fsp_err_t | RM_TOUCH_DriftControl (touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq)<br>Control drift correction. Implements touch_api_t::driftControl. More... |
| fsp_err_t | RM_TOUCH_MonitorAddressGet (touch_ctrl_t *const p_ctrl, uint8_t **pp_monitor_buf, uint8_t **pp_monitor_id, uint16_t **pp_monitor_size)<br>This function get the variable address of QE Monitor. This function is used for QE Monitor when both automatic judgement and software judgement are operated. More... |

### Detailed Description

This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch Middleware Interface.

# Overview

The Touch Middleware uses the CTSU (r_ctsu) API and provides application-level APIs for scanning touch buttons, sliders, and wheels. This module is configured via the QE for Capacitive Touch.

### Features

- Supports touch buttons (Self and Mutual), sliders, and wheels
- Can retrieve the status of up to 64 buttons at once
- Software and external triggering
- Callback on scan end
- Collects and calculates usable scan results:
  - Slider position from 1 to 100 (percent)
  - Wheel position from 1 to 360 (degrees)
- Dynamic touch-judgment-threshold adjustment
- Calculate the XY coordinates of the pad(CTSU2)
- Optional (build time) support for real-time monitoring functionality through the QE tool over UART

- Optional (build time) support for tuning function through the QE Standalone Version tool over UART
- TrustZone Support

# Configuration

*Note*

***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e² studio and search for "QE".***

This module supports the QE monitor function. The monitor determines whether to use debugger or serial communications, determines the type of the information from QE and sends only the necessary information. This module supports the serial tuning function with the standalone version of QE. Generates a configuration configuration file by UART communication with QE.

*Note*

*Multiple configurations can be defined within a single project allowing for different scan procedures or button layouts.*

### Build Time Configurations for rm_touch

The following build time configurations are defined in fsp_cfg/rm_touch_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support for QE monitoring using UART | • Enabled<br>• Disabled | Disabled | Enable UART support for QE monitoring. |
| Support for QE Tuning using UART | • Enabled<br>• Disabled | Disabled | Enable UART support for QE Tuning. |
| Type of chattering suppression | • TypeA : Counter of exceed threshold is hold within hysteresis range<br>• TypeB : Counter of exceed threshold is reset within hysteresis range. | TypeA : Counter of exceed threshold is hold within hysteresis range | TypeA of chattering suppression : Counter of exceed threshold is hold within hysteresis range. / TypeB of chattering suppression : Counter of exceed threshold is reset within hysteresis range. |

### Configurations for CapTouch > Touch (rm_touch)

This module can be added to the Stacks tab via New Stack > CapTouch > Touch (rm_touch). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|

### Interrupt Configuration

Refer to the CTSU (r_ctsu) section for details.

### Clock Configuration

Refer to the CTSU (r_ctsu) section for details.

### Pin Configuration

Refer to the CTSU (r_ctsu) section for details.

## Usage Notes

### Measurements and Data Processing

The module determines whether the button has been touched based on the change in capacitance and detects the position of the slider or wheel. This requires continued periodic measurements of capacitance. When developing your application, make sure to periodically call RM_TOUCH_ScanStart() and RM_TOUCH_DataGet(). For more details, refer to the sample application.

### Button Touch Determination

### Touch judgment type

Touch detection of each method can be set to Value Majority Mode(VMM) or Judgment Majority Mode(JMM).
Multiple methods can be configured in touch interface configurations and each method can be independently configured in VMM or JMM.

1. Value Majority Mode (VMM)
   Touch module obtains one measurement result by combining the two measured value which have less noise out of three measured value for each button from CTSU module and outputs the touch judgment result.
   As a result, the measurement results has been noise-reduced in CTSU (r_ctsu) module.
2. Judgment Majority Mode (JMM)
   This mode supported from FSP V5.6.0 or later.
   Touch module obtains three measurement results for each button from CTSU module, judges each touch, and outputs the final touch judgment result by majority vote. This mode can be only applied to a method which contains button only.

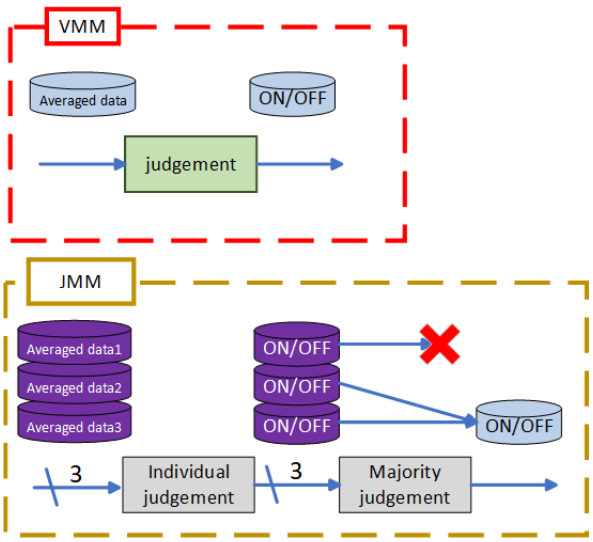



Figure 194: Data flow comparison of VMM and JMM

### Creating reference value and threshold

A touch button is not a mechanical button in which the ON/OF state is switched by hardware. The ON/OFF state is determined via software.
First, a reference value is created based on measurement results in the non-touch state. The initial reference value is the first measured value. The threshold is then determined with an arbitrary offset. If a measured value exceeds the threshold, the button is determined to be in the ON state, if it does not exceed the threshold, it is in the OFF state.
Processing for self-capacitance and mutual capacitance are basically the same. However, because the amount of capacitance decreases when a mutual capacitance button is touched, the user needs to set the threshold based on decreasing measured values to determine the ON/OFF state.
You can set the threshold for each button separately in the configuration settings (threshold in touch_button_cfg_t). The following functions are also included to deal with issues such as chattering suppression and changes in the external environment which affect actual touch recognition.
VMM : Creates one reference value and one threshold value for each button.
JMM : Creates three reference values and three threshold values for each button.

### Positive Noise Filter/Negative Noise Filter

As a chattering countermeasure, you can confirm the ON/OFF state after a set number of consecutive ON or OFF determinations.
In the configuration settings (on_freq and off_freq in touch_cfg_t) set the number of consecutive ON or OFF states. You can do this for all buttons in the touch interface configuration. Be aware that, although this is an effective solution to improving chattering, the greater the number of consecutive states, the slower the response to actual touch.

### Hysteresis

This is another chattering countermeasure. Offset the constant to the threshold after the state goes to ON, and prevent chattering by using hysteresis as the OFF-to-ON and ON-to-OFF threshold.
You can set the hysteresis value for each button in the configuration settings (hysteresis in touch_button_cfg_t). The larger the hysteresis, the more effective the countermeasure is in suppressing chattering. However, keep in mind that this will make it more difficult to return the state

from ON-to-OFF of OFF-to-ON.
VMM : One hysteresis for each button.
JMM : Three hysteresis for each button.

### Drift Correction Process

As a countermeasure for changes in the external environment, the drift correction process refreshes the reference value.
After averaging the measured value in the OFF state over a set period, if the button is in the touch OFF state after a set period, the reference value is refreshed. The drift correction is only executed in the OFF state and is cleared when touch ON is determined.
Set the period in the configuration settings (drift_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration. This allows you to adjust the ability to determine the touch state despite changes in the external environment.
VMM : Drift corrects one baseline value per button.
JMM : Drift corrects three baseline value per button.

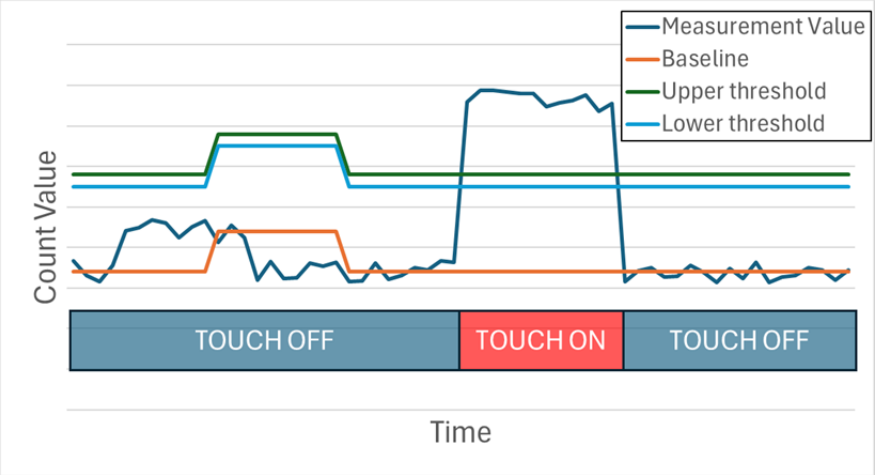



Figure 195: Button Touch Determination

### Press and hold cancel

Strong noise or other sudden environment changes can disable the drift correction process, preventing return from the ON state. The press and hold cancel function implements the drift correction process and returns the button from the ON state by forcibly turning the state to OFF after a certain number of consecutive ON state periods.
Set the number of consecutive ON periods required for the press and hold cancel function to return the button to the OFF state in the configuration settings (cancel_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration.

### Chattering suppression type (Build option)

This build option is a function to supplement the above functions (Positive Noise Filter/Negative Noise Filter and Hysteresis) for performing touch judgment.
This build option changes the processing method for Counter of exceed threshold to TypeA or TypeB.
TypeA of chattering suppression : Counter of exceed threshold is hold within hysteresis range.
TypeB of chattering suppression : Counter of exceed threshold is reset within hysteresis range.

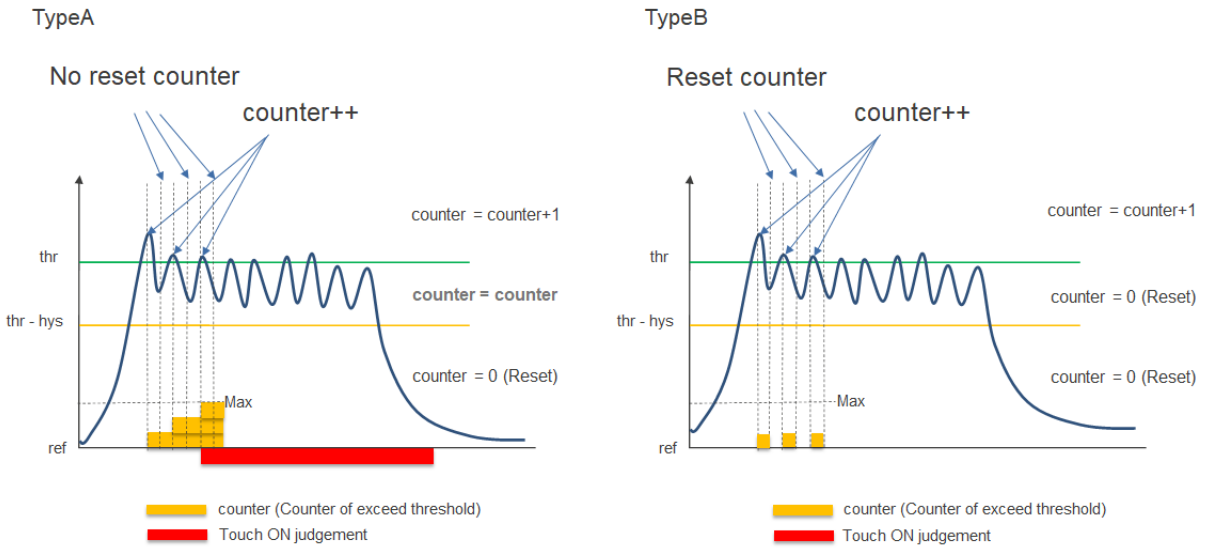



Figure 196: Chattering suppression type

### Touch Position Detection of Slider/Wheel

Configure a slider with multiple terminals to be measured (TS) physically arranged in a straight line.
Configure a wheel with multiple terminals physically arranged in a circle.
The touch position is calculated from the measured values of the TS in the configuration. The calculation method for sliders and wheels is fundamentally the same.

1. Detect the maximum value (TS_MAX) among the terminals in the configuration.
2. Calculate the difference (d1, d2) between TS_MAX and the terminals on either side. (If the TS_MAX terminal is at one end of the slider, use the values of the two terminals to the right or left, accordingly.)
3. If the total of d1 and d2 exceeds the threshold, position calculation is initiated. If the total amount does not exceed the threshold, the position calculation process is ended.
4. With TS_MAX as the middle position, the ratio of d1 to d2 is used to calculate the position. The slider has a range of 1 to 100, and the while has a range of 1 to 360.

| | Slider | Wheel |
|---|---|---|
| Electrode type | Self capacitance only | Self capacitance only |
| Number of electrodes | 3-10 | 4+ |
| Touch position output range | 1-100 | 1-360 |
| Default value (no touch) | 0xFFFF | 0xFFFF |

### Tuning the Touch Determination Adjustment

When QE tuning, a measurement is performed with a finger touching the button and the tuned parameters are output in the configuration file. The setting value of the threshold is 60% of the touch sensitivity between touch and non-touch state, and the setting value of the hysteresis coefficient is 5% of the threshold.
This module provides the functions for dynamic adjusting of these threshold and hysteresis coefficient.
They are two functions as below.

**Adjusting the threshold and hysteresis coefficient to an arbitrary ratio.**
**Use RM_TOUCH_ThresholdAdjust().**

When changing the touch determination threshold ratio from 60% QE set to 70% user specified, the touch determination thresholds are as below.
If you want to make this setting, set the member of the second argument as follows. It is also necessary to set the ratio of the amount of touch change and the hysteresis value.

*p_touch_sensitivity_ratio = 100,
old_threshold_ratio = 60,
new_threshold_ratio = 70,
new_hysteresis_ratio = 5

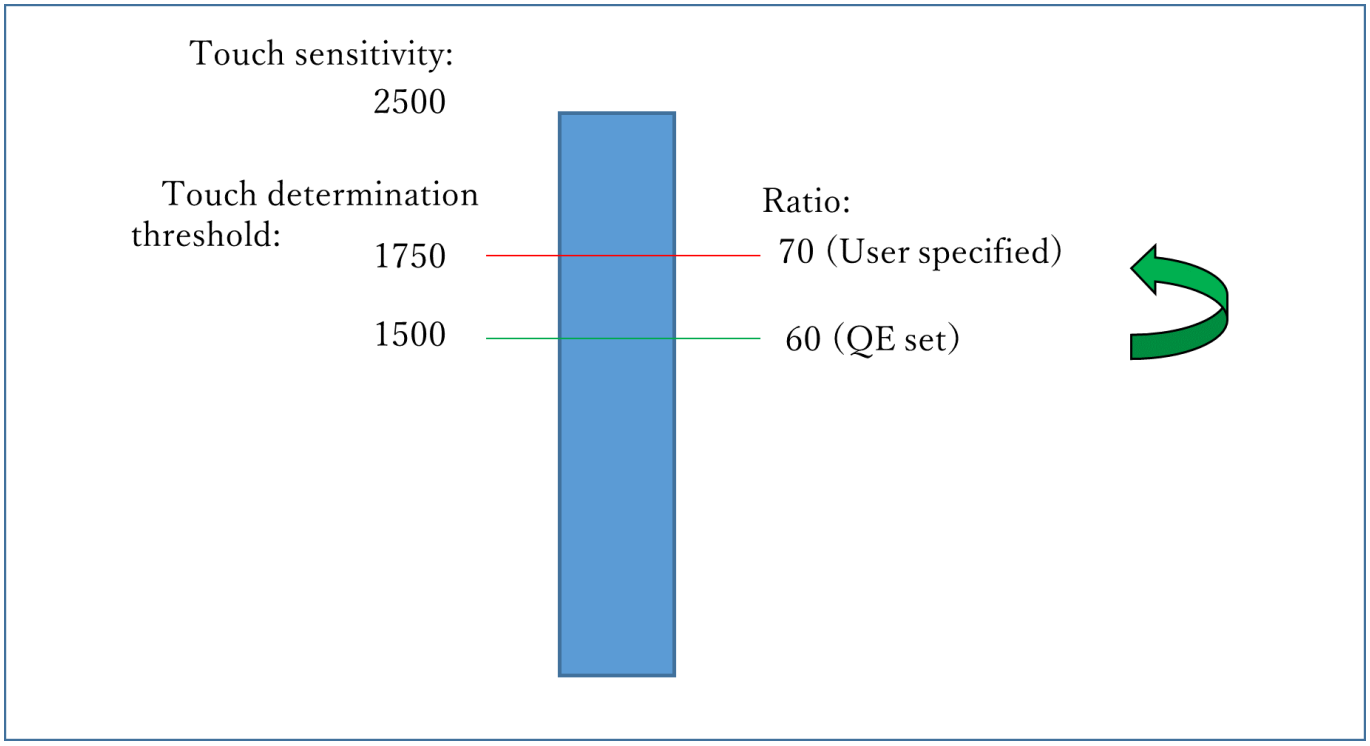



Figure 197: Example of changing the threshold ratio

**Adjusting the threshold and hysteresis coefficient according to the current touch sensitivity.**
**Use RM_TOUCH_SensitivityRatioGet(), RM_TOUCH_ThresholdAdjust(), and RM_TOUCH_DriftControl().**

When changing the kind of the overlay panel, the touch sensitivity differs from the one QE tuned. Wanting to use the software as it is without re-tuning. If you use a thicker overlay than that at QE tuning, the touch sensitivity decreases, and a touch may not be determined because of the same touch determination threshold. This function adjusts the touch determination threshold based on the ratio of the touch sensitivity after changing the overlay to the touch sensitivity at the QE tuning.

RM_TOUCH_SensitivityRatioGet() outputs the ratio of the current touch sensitivity assuming that the touch sensitivity at the QE setting is 100%.
The following figure shows the case where an overlay panel is thinner and the touch sensitivity increases.

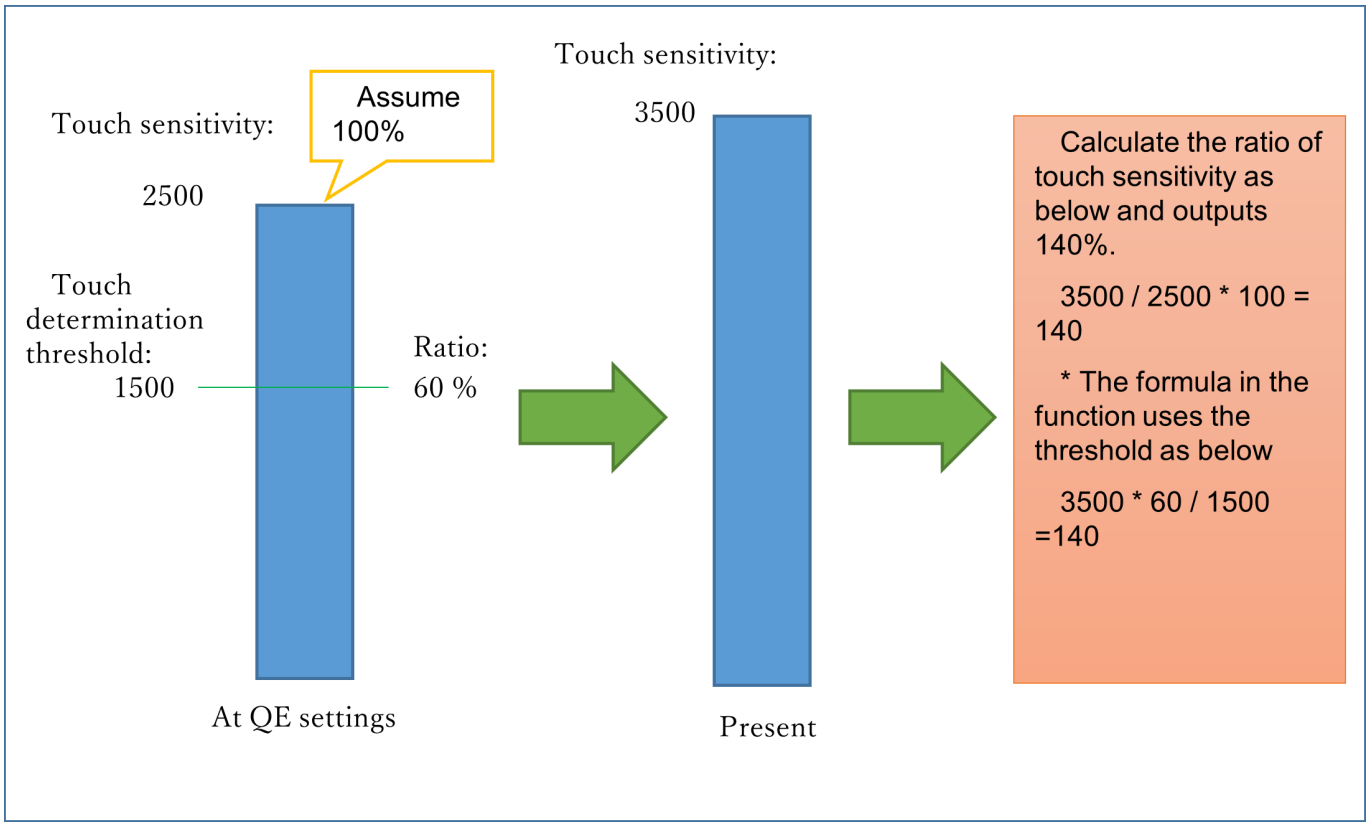



Figure 198: Example of increase touch sensitivity for thin overlay panels

Following figure shows the case where an overlay panel is thicker and the touch sensitivity decreases.

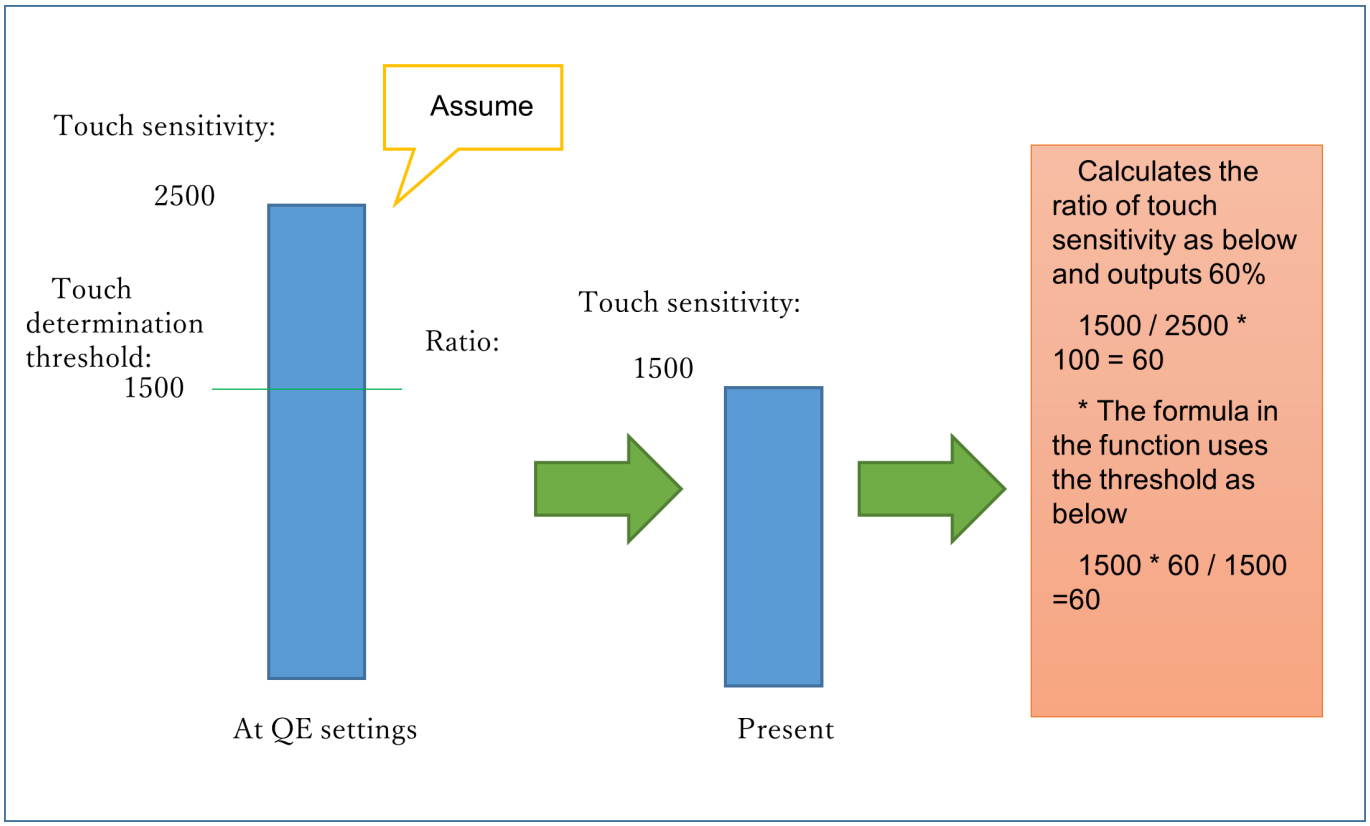



Figure 199: Example of decrease touch sensitivity for thicker overlay panels

RM_TOUCH_ThresholdAdjust() sets the new touch determination threshold and the hysteresis value by using the touch sensitivity ratio obtained with RM_TOUCH_SensitivityRatioGet() as arguments.

Example of calculation 1:
The touch sensitivity ratio is 140%, and the threshold set by QE is 1500.
Threshold = 140 * 1500 / 100 = 2100

*p_touch_sensitivity_ratio = 140,
old_threshold_ratio = 60,
new_threshold_ratio = 60,
new_hysteresis_ratio = 5

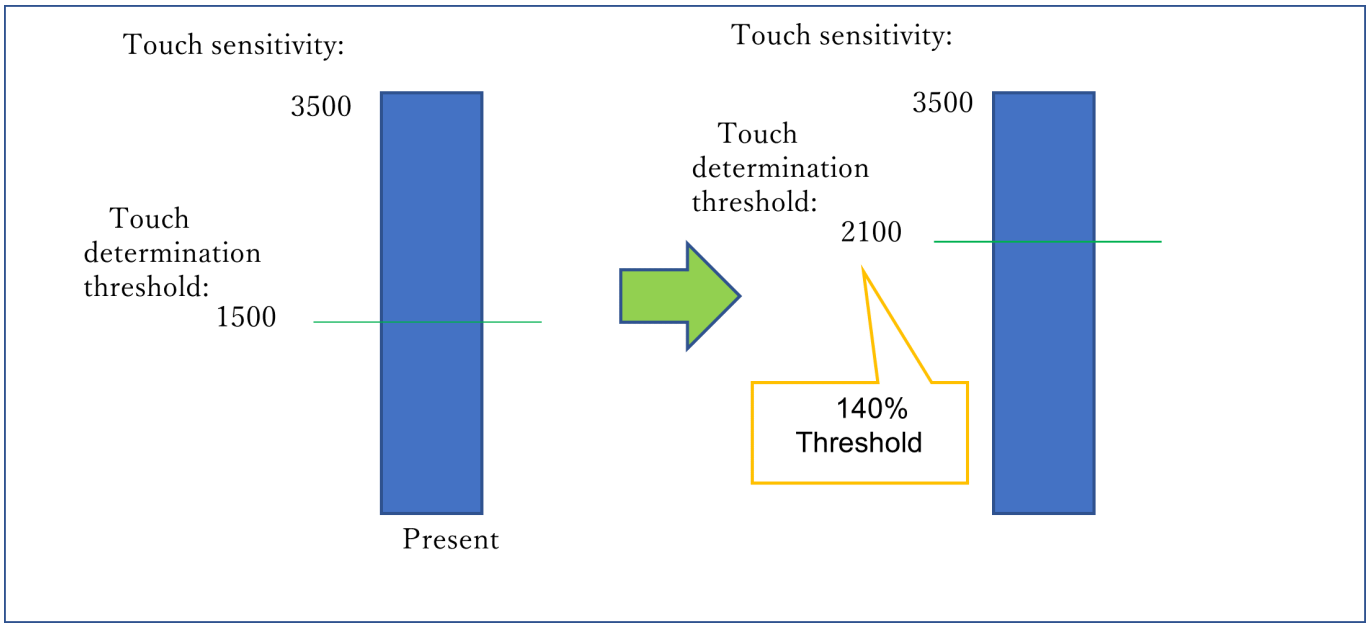



Figure 200: Example of calculation 1

Example of calculation 2:
The touch sensitivity ratio is 60%, and the threshold set by QE is 1500.
Threshold = 60 * 1500 / 100 = 900

*p_touch_sensitivity_ratio = 60,
old_threshold_ratio = 60,
new_threshold_ratio = 60,
new_hysteresis_ratio = 5

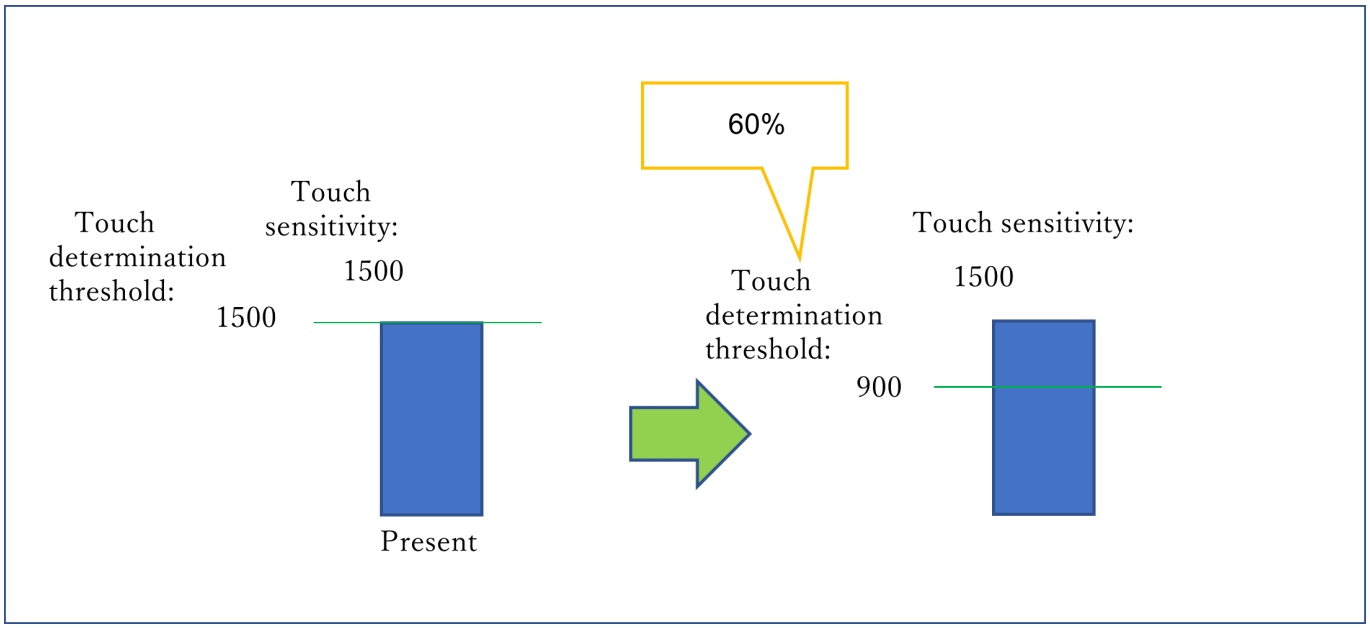



Figure 201: Example of calculation 2

RM_TOUCH_DriftContorol() set the second argument to 0 to stop the drift correction function.

When calculating the ratio of the touch change amount using RM_TOUCH_SensitivityRatioGet(), the touch change amount decreases due to the thick overlay, and the threshold value is not exceeded even if touched. Prevents the reference value from drifting.

**Example of the application for adjustment using data flash without re-tuning or software rewriting**

Enable UART communication to PC and 'tuning mode'. In tuning mode, the MCU transmits the ratio of the touch sensitivity in the touch state to the PC in real time. A user sends a command to decide the ratio while monitoring on the PC. The MCU stores the received ratio in the data flash. Make sure that the ratio stored in the data flash is read at the software activation, and the touch determination threshold is adjusted based on this stored value.

**Pad**

Configure a pad with multiple terminals physically arranged in cross.
The current position is Calculated from the measured values of the CTSU mutual scanning in the configuration.
Use RM_TOUCH_PadDataGet().
Pad is subject so some limitations:

| | Pad |
|---|---|
| Electrode type | CFC mutual capacitance only |
| Number of electrodes | RX(TS-CFC)3+, TX(Any TS)3+ |
| Touch position output range | rx_coodinate:(0 ~ rx_pixel), tx_coodinate:(0 ~ tx_pixsel) |
| Default value (no touch) | rx_coodinate:0xFFFF, tx_coodinate:0xFFFF |
| Pixel range | rx_pixel:(1 ~ 65535), tx_pixsel:(1 ~ 65535) |

Pitch for each terminal can be set with QE. Pitch's default value is 64.
The relationship between pixel and pitch : Pixel = Pitch × number of TS - 1

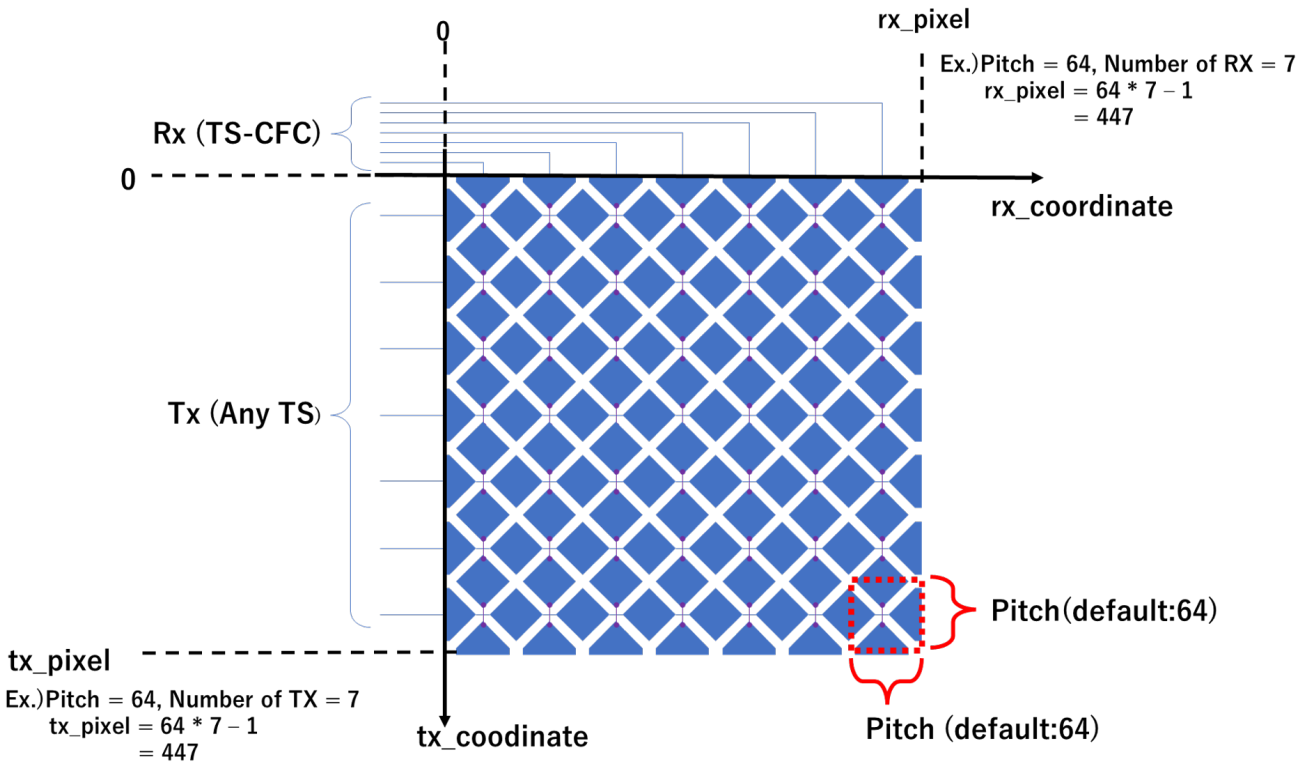



Figure 202: Example of Pad

### TrustZone Support

In r_ctsu and rm_touch module, Non-Secure Callable Guard Functions are only generated from QE for Capacitive Touch. QE can be used for tuning in secure or flat project, but not in non-secure project. If you want to use in non-secure project, copy the output file from secure or flat project. Refer to QE Help for more information.

## Examples

### Basic Example

This is a basic example of minimal use of the TOUCH in an application.

```
void touch_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
```

```
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
       g_flag = 0;
       err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
       }
    }
}
```

### Multi Mode Example

This is a optional example of using both Self-capacitance and Mutual-capacitance. Refer to the Multi Mode Example in CTSU usage notes.

```
void touch_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
    assert(FSP_SUCCESS == err);
    err = RM_TOUCH_Open(&g_touch_ctrl_mutual, &g_touch_cfg_mutual);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
       g_flag = 0;
 RM_TOUCH_ScanStart(&g_touch_ctrl_mutual);
 while (0 == g_flag)
```

```
     {
/* Wait scan end callback */
      }
      g_flag = 0;
      err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
      err = RM_TOUCH_DataGet(&g_touch_ctrl_mutual, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | touch_button_info_t |
| struct | touch_slider_info_t |
| struct | touch_wheel_info_t |
| struct | touch_pad_info_t |
| struct | touch_mm_info_t |
| struct | touch_instance_ctrl_t |

## Data Structure Documentation

### ◆ touch_button_info_t

| struct touch_button_info_t | | |
|---|---|---|
| Information of button | | |
| Data Fields | | |
| uint64_t | status | Touch result bitmap. |
| uint16_t * | p_threshold | Pointer to Threshold value array. |

| | | |
|---|---|---|
| | | g_touch_button_threshold[] is set by Open API. |
| uint16_t * | p_hysteresis | Pointer to Hysteresis value array. g_touch_button_hysteresis[] is set by Open API. |
| uint16_t * | p_reference | Pointer to Reference value array. g_touch_button_reference[] is set by Open API. |
| uint16_t * | p_on_count | Continuous touch counter. g_touch_button_on_count[] is set by Open API. |
| uint16_t * | p_off_count | Continuous non-touch counter. g_touch_button_off_count[] is set by Open API. |
| uint32_t * | p_drift_buf | Drift reference value. g_touch_button_drift_buf[] is set by Open API. |
| uint16_t * | p_drift_count | Drift counter. g_touch_button_drift_count[] is set by Open API. |
| uint8_t | on_freq | Copy from config by Open API. |
| uint8_t | off_freq | Copy from config by Open API. |
| uint16_t | drift_freq | Copy from config by Open API. |
| uint16_t | cancel_freq | Copy from config by Open API. |

### ◆ touch_slider_info_t

| struct touch_slider_info_t | | |
|---|---|---|
| Information of slider | | |
| Data Fields | | |
| uint16_t * | p_position | Calculated Position data. g_touch_slider_position[] is set by Open API. |
| uint16_t * | p_threshold | Copy from config by Open API. g_touch_slider_threshold[] is set by Open API. |

### ◆ touch_wheel_info_t

| struct touch_wheel_info_t | | |
|---|---|---|
| Information of wheel | | |
| Data Fields | | |
| uint16_t * | p_position | Calculated Position data. |

| | | |
|---|---|---|
| | | g_touch_wheel_position[] is set by Open API. |
| uint16_t * | p_threshold | Copy from config by Open API. g_touch_wheel_threshold[] is set by Open API. |

### ◆ touch_pad_info_t

| struct touch_pad_info_t | | |
|---|---|---|
| Information of pad | | |
| Data Fields | | |
| uint16_t * | p_rx_coordinate | RX coordinate. |
| uint16_t * | p_tx_coordinate | TX coordinate. |
| uint16_t * | p_num_touch | number of touch |
| uint16_t * | p_threshold | Coordinate calculation threshold value. |
| uint16_t * | p_base_buf | ScanData Base Value Buffer. |
| uint16_t * | p_rx_pixel | X coordinate resolution. |
| uint16_t * | p_tx_pixel | Y coordinate resolution. |
| uint8_t * | p_max_touch | Maximum number of touch judgments used by the pad. |
| int32_t * | p_drift_buf | Drift reference value. g_touch_button_drift_buf[] is set by Open API. |
| uint16_t * | p_drift_count | Drift counter. g_touch_button_drift_count[] is set by Open API. |
| uint8_t | num_drift | Copy from config by Open API. |

### ◆ touch_mm_info_t

| struct touch_mm_info_t |
|---|
| Information of touch button judge(CTSU2) |

### ◆ touch_instance_ctrl_t

| struct touch_instance_ctrl_t | | |
|---|---|---|
| TOUCH private control block. DO NOT MODIFY. Initialization occurs when RM_TOUCH_Open() is called. | | |
| Data Fields | | |
| uint32_t | open | Whether or not driver is open. |
| touch_button_info_t | binfo | Information of button. |

| | | |
|---|---|---|
| touch_slider_info_t | sinfo | Information of slider. |
| touch_wheel_info_t | winfo | Information of wheel. |
| bool | serial_tuning_enable | Flag of serial tuning status. |
| touch_pad_info_t | pinfo | Information of pad. |
| touch_cfg_t const * | p_touch_cfg | Pointer to initial configurations. |
| ctsu_instance_t const * | p_ctsu_instance | Pointer to CTSU instance. |
| touch_mm_info_t * | p_touch_mm_info | Pointer to information of touch button judge. |

### Function Documentation

#### ◆ RM_TOUCH_Open()

fsp_err_t RM_TOUCH_Open ( touch_ctrl_t *const *p_ctrl*, touch_cfg_t const *const *p_cfg* )

Opens and configures the TOUCH Middle module. Implements touch_api_t::open.

Example:

```
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | TOUCH successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_TOUCH_ScanStart()

fsp_err_t RM_TOUCH_ScanStart ( touch_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance or other. |
| FSP_ERR_CTSU_NOT_GET_DATA | The previous data has not been retrieved by DataGet. |

### ◆ RM_TOUCH_DataGet()

fsp_err_t RM_TOUCH_DataGet ( touch_ctrl_t *const *p_ctrl*, uint64_t * *p_button_status*, uint16_t * *p_slider_position*, uint16_t * *p_wheel_position* )

Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet.

***Note***

*FSP v4.0.0 or later,*

- *The value of 'Secondary - Primary' is modified from uint16_t to int16_t. When the value of 'Secondary - Primary' is larger than 32767 and less than -32767, this API return FSP_ERR_INVALID_DATA.*
- *An upper limit is set for the value of Secondary. When the value of Secondary is larger than 45000, this API return FSP_ERR_INVALID_DATA.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_DATA | Accuracy of data is not guaranteed. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ RM_TOUCH_PadDataGet()

fsp_err_t RM_TOUCH_PadDataGet ( touch_ctrl_t *const *p_ctrl*, uint16_t * *p_pad_rx_coordinate*, uint16_t * *p_pad_tx_coordinate*, uint8_t * *p_pad_num_touch* )

This function gets the current position of pad is being pressed. Implements touch_api_t::padDataGet , g_touch_on_ctsu.

***Note***

*FSP v4.0.0 or later,*

- *The value of 'Secondary - Primary' is modified from uint16_t to int16_t. When the value of 'Secondary - Primary' is larger than 32767 and less than -32767, this API return FSP_ERR_INVALID_DATA.*
- *An upper limit is set for the value of Secondary. When the value of Secondary is larger than 45000, this API return FSP_ERR_INVALID_DATA.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_DATA | Accuracy of data is not guaranteed. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |

### ◆ RM_TOUCH_ScanStop()

fsp_err_t RM_TOUCH_ScanStop ( touch_ctrl_t *const *p_ctrl*)

Scan stop specified TOUCH control block. Implements touch_api_t::scanStop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_TOUCH_CallbackSet()

fsp_err_t RM_TOUCH_CallbackSet ( touch_ctrl_t *const *p_api_ctrl*, void(*)(touch_callback_args_t *) *p_callback*, void *const *p_context*, touch_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements touch_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ RM_TOUCH_Close()

fsp_err_t RM_TOUCH_Close ( touch_ctrl_t *const *p_ctrl*)

Disables specified TOUCH control block. Implements touch_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_TOUCH_SensitivityRatioGet()

fsp_err_t RM_TOUCH_SensitivityRatioGet ( touch_ctrl_t *const *p_ctrl*, touch_sensitivity_info_t * *p_touch_sensitivity_info* )

Get the touch sensitivity ratio. Implements touch_api_t::sensitivityRatioGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully touch sensitivity ratio got. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CTSU_SCANNING | Scanning this instance. |
| FSP_ERR_CTSU_INCOMPLETE_TUNING | Incomplete initial offset tuning. |

### ◆ RM_TOUCH_ThresholdAdjust()

fsp_err_t RM_TOUCH_ThresholdAdjust ( touch_ctrl_t *const *p_ctrl*, touch_sensitivity_info_t * *p_touch_sensitivity_info* )

Adjust the touch judgment threshold. Implements touch_api_t::thresholdAdjust.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully touch judgment threshold was adjusted. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_TOUCH_DriftControl()

fsp_err_t RM_TOUCH_DriftControl ( touch_ctrl_t *const *p_ctrl*, uint16_t *input_drift_freq* )

Control drift correction. Implements touch_api_t::driftControl.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully drift correction was controlled. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_TOUCH_MonitorAddressGet()

fsp_err_t RM_TOUCH_MonitorAddressGet ( touch_ctrl_t *const *p_ctrl*, uint8_t ** *pp_monitor_buf*, uint8_t ** *pp_monitor_id*, uint16_t ** *pp_monitor_size* )

This function get the variable address of QE Monitor. This function is used for QE Monitor when both automatic judgement and software judgement are operated.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully QE monitor variable address was got. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened. |
| FSP_ERR_NOT_ENABLED | Requested operation is not enabled. |

## 5.2.6 Connectivity

Modules

### Detailed Description

Connectivity Modules.

### Modules

| | |
|---|---|
| | Azure RTOS USBX Port (rm_usbx_port) |
| | CAN (r_can)<br>Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface. |
| | CAN FD (r_canfd)<br>Driver for the CANFD peripheral on RA MCUs. This module implements the CAN Interface. |
| | CEC (r_cec)<br>Driver for the CEC peripheral on RA MCUs. This module implements the CEC Interface. |
| | I2C Communication Device (rm_comms_i2c)<br>Middleware to implement the I2C communications interface. This module implements the Communications Middleware Interface. |
| | I2C Master (r_iic_b_master)<br>I2C Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Master (r_iic_master)<br>Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Master (r_iica_master)<br>Driver for the IICA peripheral on RA MCUs. This module implements the I2C Master Interface. |

| | |
|---|---|
| | I2C Master (r_sau_i2c)<br>Driver for the SAU peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Master (r_sci_b_i2c)<br>Driver for the SCI_B peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Master (r_sci_i2c)<br>Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface. |
| | I2C Slave (r_iic_b_slave)<br>Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Slave Interface. |
| | I2C Slave (r_iic_slave)<br>Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface. |
| | I2C Slave (r_iica_slave)<br>Driver for the IICA peripheral on RA MCUs. This module implements the I2C Slave Interface. |
| | I2S (r_ssi)<br>Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface. |
| | I3C (r_i3c)<br>Driver for the I3C peripheral on RA MCUs. This module implements the I3C Interface. |
| | LIN (r_sau_lin)<br>LIN driver for the SAU peripheral on RA MCUs. This module implements the LIN Interface. |
| | LIN (r_sci_b_lin) |

| | |
|---|---|
| | Driver for the SCI peripheral on RA MCUs. This module implements the LIN Interface. |
| | LIN (r_sci_lin)<br>Driver for the SCI peripheral on RA MCUs. This module implements the LIN Interface. |
| | SMBUS Communication Device (rm_comms_smbus)<br>Middleware to implement the SMBUS communications interface. This module implements the Communications Middleware Interface. |
| | SMCI (r_sci_smci)<br>Driver for the SCI peripheral on RA MCUs. This module implements the SMCI Interface. |
| | SPI (r_sau_spi)<br>Driver for the SAU peripheral on RA MCUs. This module implements the SPI Interface. |
| | SPI (r_sci_b_spi)<br>Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface. |
| | SPI (r_sci_spi)<br>Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface. |
| | SPI (r_spi)<br>Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface. |
| | SPI (r_spi_b)<br>Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface. |
| | SPI Communication Device (rm_comms_spi)<br>Middleware to implement a generic communications interface over |

| | |
|---|---|
| | SPI. This module implements the Communications Middleware Interface. |
| | UART (r_sau_uart)<br>UART driver for the SAU peripheral on RA MCUs. This module implements the UART Interface. |
| | UART (r_sci_b_uart)<br>Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface. |
| | UART (r_sci_uart)<br>Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface. |
| | UART (r_uarta)<br>Driver for the UARTA peripheral on RA MCUs. This module implements the UART Interface. |
| | UART Communication Device (rm_comms_uart)<br>Middleware to implement a generic communications interface over UART. This module implements the Communications Middleware Interface. |
| | USB (r_usb_basic)<br>Driver for the USB peripheral on RA MCUs. This module implements the USB Interface. |
| | USB Composite (r_usb_composite) |
| | USB HAUD (r_usb_haud)<br>This module provides a USB Peripheral Audio Device Class Driver (HAUD). It implements the USB HAUD Interface. |
| | USB HCDC (r_usb_hcdc)<br>This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface. |
| | |

| | |
|---|---|
| | USB HCDC_ECM (r_usb_hcdc_ecm)<br><br>This module provides a USB Host Communications Device Class (HCDC-ECM) driver. It implements the USB HCDC_ECM (r_usb_hcdc_ecm). |
| | USB HHID (r_usb_hhid)<br><br>This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface. |
| | USB HMSC (r_usb_hmsc)<br><br>This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface. |
| | USB Host Vendor class (r_usb_hvnd) |
| | USB PAUD (r_usb_paud)<br><br>This module provides a USB Peripheral Audio Device Class Driver (PAUD). It implements the USB PAUD Interface. |
| | USB PCDC (r_usb_pcdc)<br><br>This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface. |
| | USB PHID (r_usb_phid)<br><br>This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface. |
| | USB PMSC (r_usb_pmsc)<br><br>This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface. |
| | USB PPRN (r_usb_pprn)<br><br>This module is USB Peripheral Printer Device Class Driver (PPRN). It implements the USB PPRN Interface. |
| | USB Peripheral Vendor class (r_usb_pvnd) |
| | USB-TYPEC (r_usb_typec) |

| | |
|---|---|
| | Driver for the USB_TYPEC peripheral on RA MCUs. This module implements the USB TYPEC Interface. |
| | USB_PCDC Communication Device (rm_comms_usb_pcdc)<br><br>Middleware to implement a generic communications interface over USB_PCDC. This module implements the Communications Middleware Interface. |

#### 5.2.6.1 Azure RTOS USBX Port (rm_usbx_port)

Modules » Connectivity

**Functions**

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

# Overview

This USB driver works by combining USBX and r_usb_basic module.

# How to Configuration

**Using a class other than Composite(Pereipheral), HMSC and OTG.**

The following describes how to configure USBX using PCDC as an example.

- Select [New Stack]->[Connectivity]->[Azure RTOS PCDC]

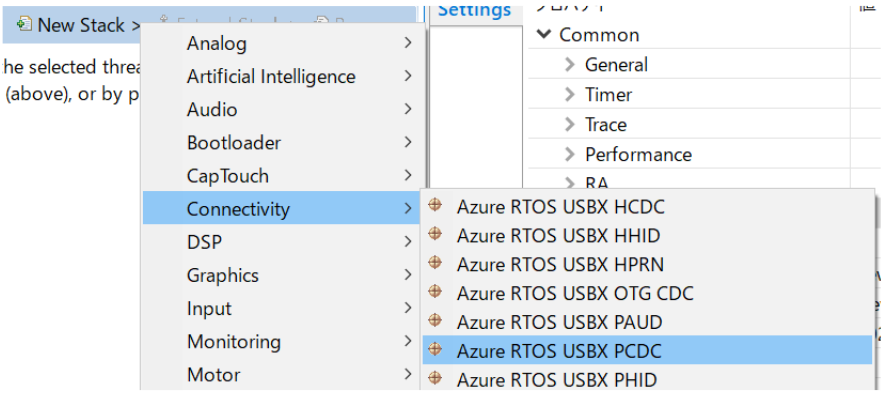



Figure 203: Select USB Device Class

**Using Composite (Peripheral)**

- Select [New Stack]->[Connectivity]->[USB Composite]

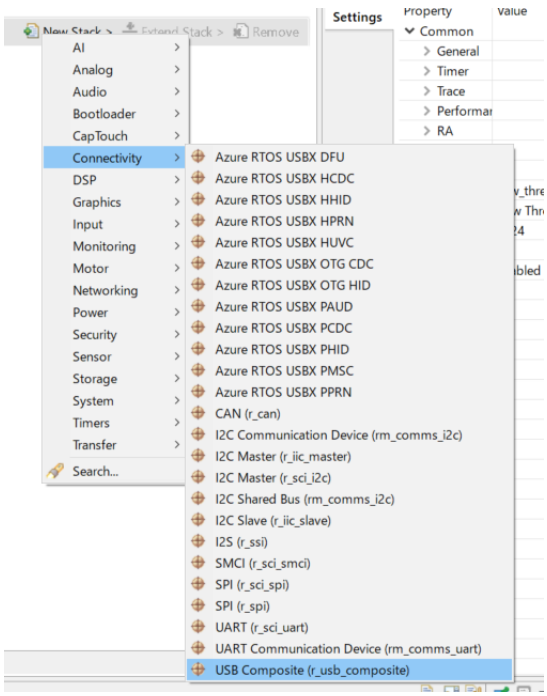



Figure 204: Select USB Composite

- The following is displayed when selecting [USB Composite].

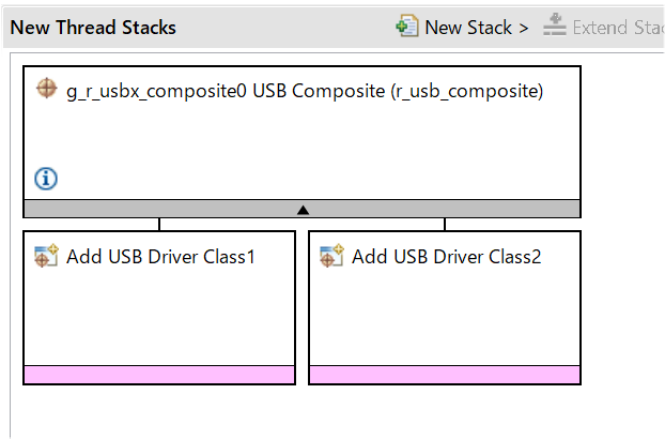



Figure 205: USB Composite Stack

- Select the supported 2 device classes as follows.

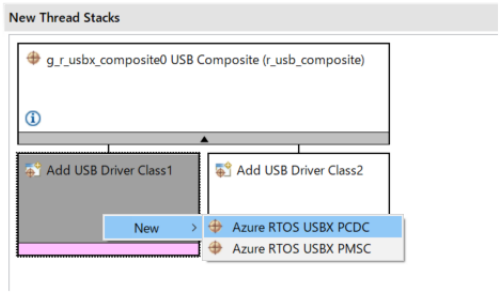



Figure 206: Select Device Classes

- Delete the "g_basic1" instance manually.

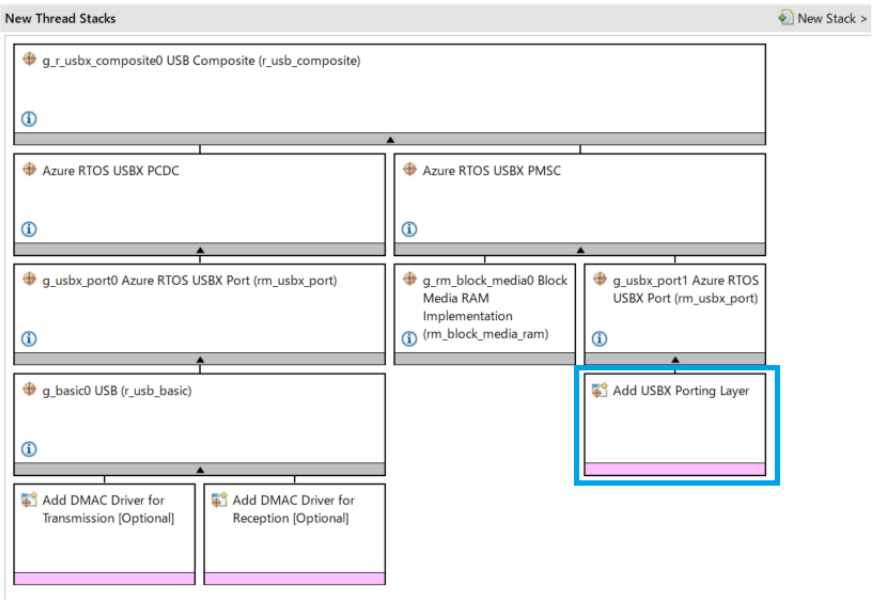



Figure 207: Delete USB Basic instance

### Using HMSC.

Since HMSC is used in a different way from other USBX modules, the usage is described below.

- Select [New Stack]->[Storage]->[Azure RTOS FileX on USBX]

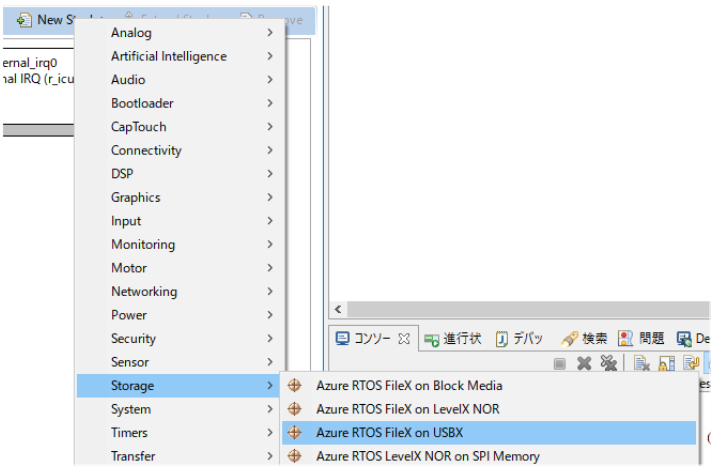



Figure 208: Select USB Device Class

- The following is displayed when selecting Filex on USBX.

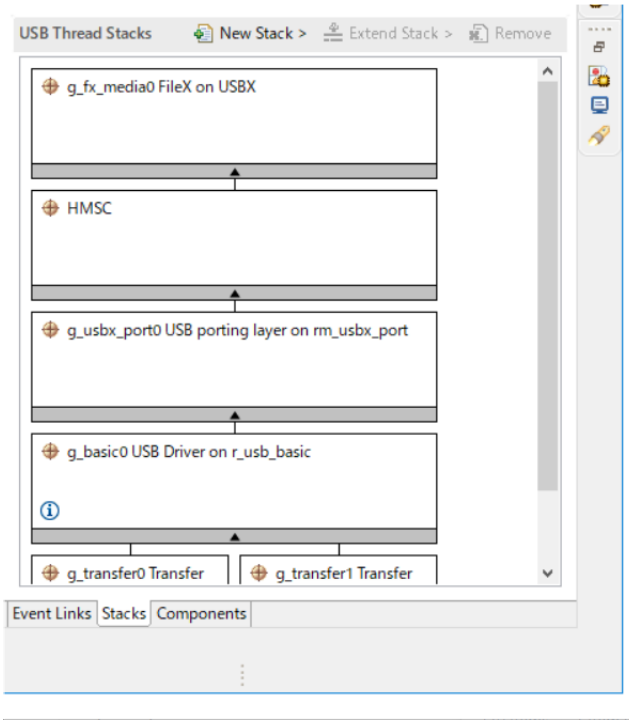



Figure 209: FileX on USBX Stack

### Using OTG.

The following describes how to configure USBX OTG.

- When using CDC / HID class
  - [New Stack]->[Connectivity]->[Azure RTOS USBX OTG CDC] or [Azure RTOS USBX OTG HID])
- When using MSC class
  - [New Stack]->[Storage]->[Azure RTOS FileX on USBX])
  - [New]->[Azure RTOS USBX OTG MSC]

- Select [New Stack]->[Input] -> [External IRQ(r_icu)]

  Be sure to select "r_icu" in OTG. The "r_icu" is used to detect the attaching of A cable.

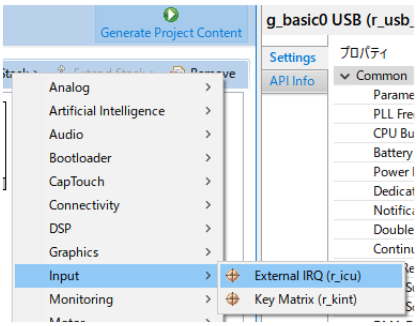



Figure 210: Select USB Device Class for OTG

- Be sure to set the following to each item in "r_icu".
  - Set "7" to "Channel" item.
  - Set "Both Edges" to "Trigger" item.
  - Set "Enabled" to "Digital Filtering" item.
  - Set "usb_otg_irq_callback" to "Callback" item.

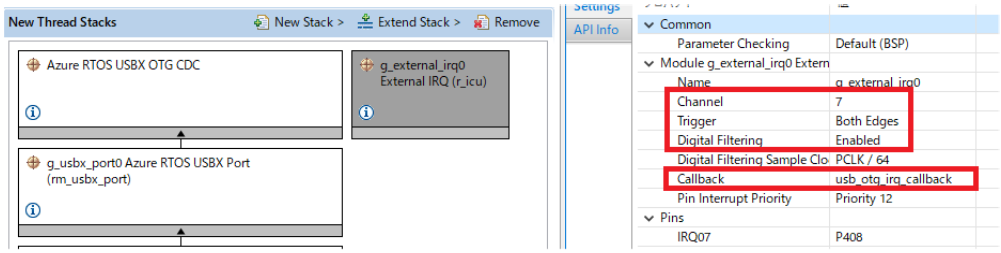



Figure 211: Select USB Device Class for OTG

- Be sure to set the following to each item in "r_usb_basic".
  - Set "Full Speed" to "USB Speed" item.
  - Set "USB_IP0 Port" to "USB Module Number".
  - Set "Not Supported" to "USBFS Resume Priority".

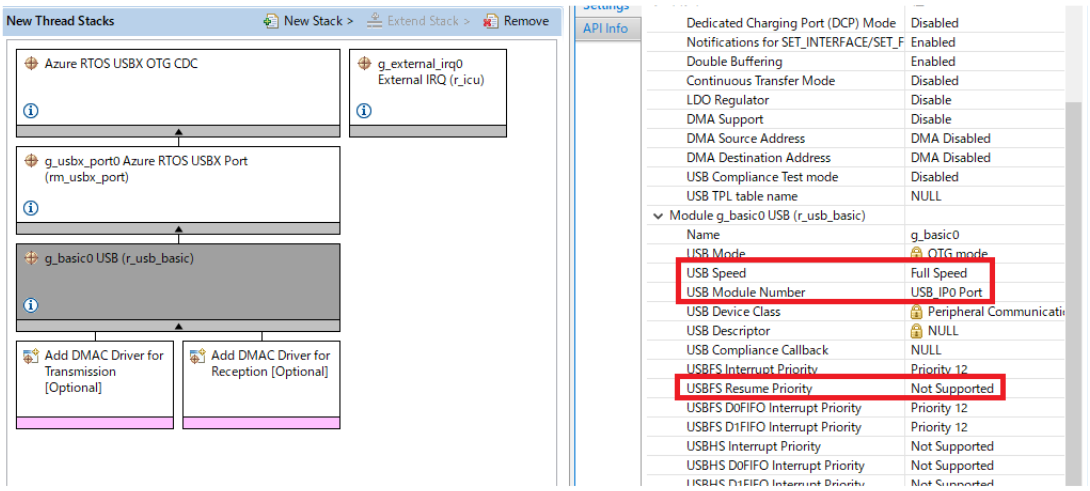



Figure 212: Select USB Device Class

- Set the following pins for USB (r_usb_basic)
  - VBUSEN
  - VBUS
  - EXICEN

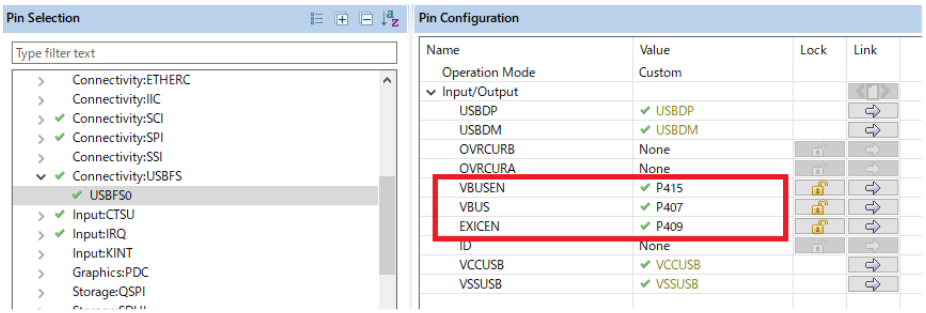



Figure 213: USB Pin Setting

- Set the following pins for IRQ (r_icu)
  - IRQ07

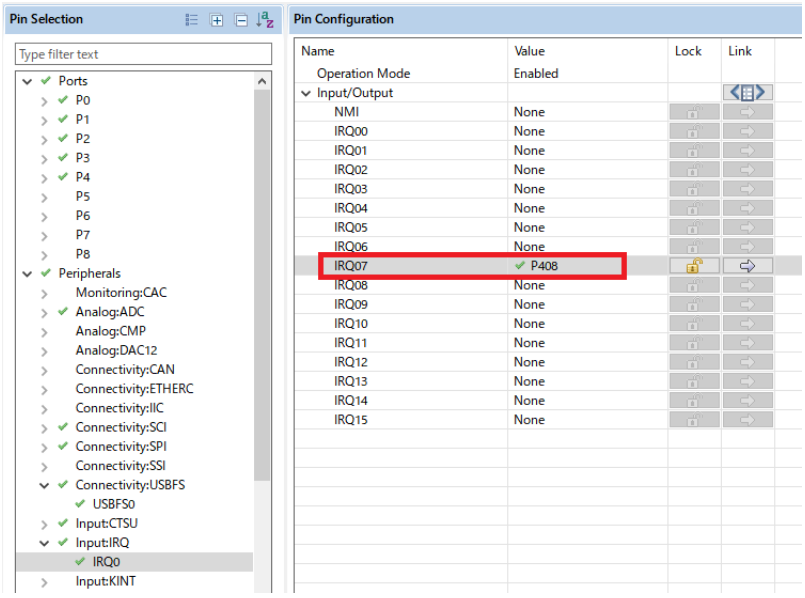



Figure 214: IRQ Pin Setting

## Using Hub

- When the user uses the following device class, the user can use USB Hub.
  - Host Communication Device Class (HCDC)
  - Host Human Interface Device Class (HHID)
  - Host Mass Storage Class (HMSC)
  - Host Printer Class (HPRN)
- Also be sure to set Enable using HUB.

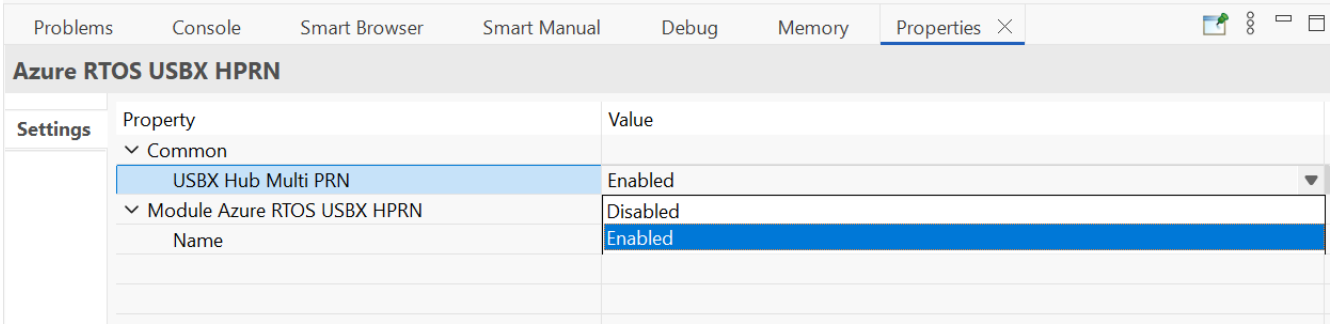



Figure 215: Enable Using Hub PRN-PRN

## Note

- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.
- The following are notes on using the ux_host_class_cdc_acm_read function or the ux_device_class_cdc_acm_read function
  - Please specify a multiple of MaxPacketSize to the 3rd argument(requested_length) since if the value of the 3rd argument is not multiple of MaxPacketSize, all data sent by USB Host or USB Peripheral may not be received correctly.
  - Please specify start address of the area allocated a size larger than 3rd argument(requested_length) to the 2nd argument(data_pointer or buffer).

- There is no stack for the Hub; please select the stack for USB Host even when using the Hub.
- Depending on the Hub used, it may be necessary to supply power to the Hub.
- HUVC does not support USB Hub.

**HMSC uses FileX.**

For more information on FileX, please refer to the following URL.

https://docs.microsoft.com/en-us/azure/rtos/filex/

**OTG**

- Be able to set UX_OTG_HOST_REQUEST_FLAG in USB Peripheral mode or call ux_host_stack_role_swap() function in USB Host mode when data other than control transfer is not being transferred.
- Register the callback function for the application using R_USB_OtgCallbackSet() function. This callback function is called when switching the operation mode(UX_OTG_MODE_IDLE/UX_OTG_MODE_SLAVE/UX_OTG_MODE_HOST).
- Use R_USB_OtgSRP() function when doing SRP(Session Request Protocol). The R_USB_OtgSRP() function does not support VBUS pulsing.
- After starting up MCU, USB is initialized as USB peripheral mode until connecting A cable.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.
- When A cable is connected, USB module is initialized as USB Host mode.
- Call R_ICU_ExternalIrqOpen() and R_ICU_ExternalIrqEnable() function in the application program since the IRQ interrupt is used to detect attaching or detaching of A cable.
- Add the OTG descriptor in the configuration descriptor in each descriptor file.

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | bLength | 1 | 5 | Size of Descriptor |
| 1 | bDescriptor Type | 1 | 9 | OTG type = 9 |
| 2 | bmAttribute | 1 | 3 or 2 | D1: HNP support<br>D0: SRP support |
| 3 | bcdOTG | 2 | 0x200 | Binary Code Descimal |

**Known Issues**

- There is compatibility issue for FileX on USBX stack in FSP 5.0.0 while importing from FSP 4.6.0 or earlier. Please refer to "GitHub Issue" about this workaround.

**Limitations**

- This USB driver does not support the multiple device class in Host mode.

- Please call the initialization function in the application program.

  Please be sure to call R_USB_Open() function after calling the following functions.

  - Peripheral

        - PCDC / PHID / PAUD / PPRN / DFU / OTG
            - ux_system_initialize
            - ux_device_stack_initialize
            - ux_device_stack_class_register
        - PMSC
            - ux_system_initialize
            - ux_device_stack_initialize
    - Host
        - HCDC / HHID / HPRN / HUVC / HAUD / OTG
            - ux_system_initialize
            - ux_host_stack_initialize
        - HMSC
            - ux_system_initialize
            - ux_host_stack_initialize
            - fx_system_initialize
- Set the value for 1000 Ticks per second in the "Timer Ticks Per Second" item.

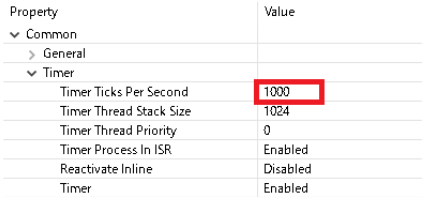



Figure 216: Specify value of Timer Ticks Per Second

- Set the Thread priority to a value of 21.

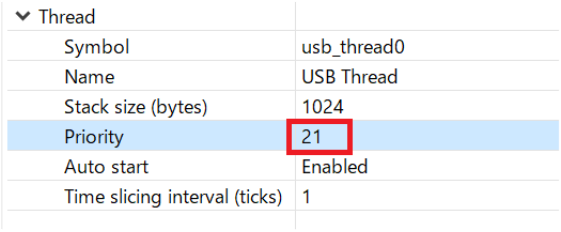



Figure 217: Specify value of Thread priority

- Call the following functions in the following order after calling the R_USB_Close() function.
    - Peripheral
        - PCDC / PHID / PPRN
            - ux_device_stack_class_unregister
            - ux_device_stack_uninitialize
        - PMSC
            - ux_device_stack_uninitialize
    - Host
        - ux_host_stack_uninitialize
- USBX Composite (Peripheral) supports the following composite device.
    - PCDC + PMSC
- USBX Composite (PCDC + PMSC) has the following notes.
    - When the user select "Safely Remove Hardware and Eject Media" on Windows and eject the mass storage (PMSC), the Windows Explorer for PMSC does not disappear. But PMSC driver works properly.

    - PMSC may not work properly when USBX Composite Device(PCDC+PMSC) is connected to linux machine (USB Host).If the user encounter this issue, please update Linux OS.
- The FAT file format that HMSC supports is FAT32, FAT16 and FAT12.
- When using USBX HMSC, please be sure to check the "fx" checkbox(red frame) of the pack the user is using in the "Components" tab(green frame) as shown below.

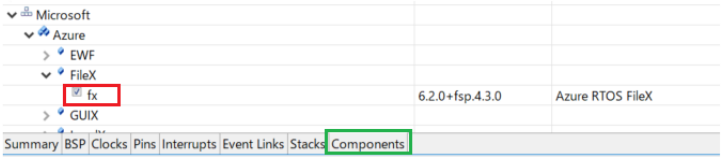
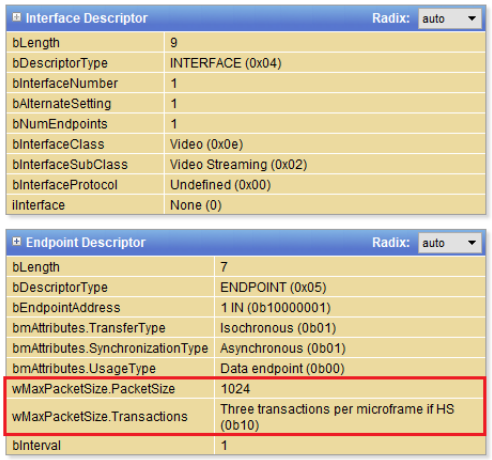



Figure 218: Check the fx checkbox

- When using USBX HPRN, if the internal buffer of the printer is full, USB data transfer speed drops significantly.
- When using USBX HHID, please use Wired device.
- USBX HUVC driver does not support Full-speed.
- USBX HUVC driver does not support the following APIs.
    - ux_host_class_video_transfer_buffers_add
    - ux_host_class_video_read
- Note the following when using *ux_host_class_video_transfer_buffer_add* function (USBX HUVC API)
    - Please call *ux_host_class_video_transfer_buffer_add* function after cofirming that the callback function for data reception completion is called when calling this API again. This callback function is registered by *ux_host_class_video_transfer_callback_set* function in the application program.
    - Please specify a 4-byte alignment address for the 2nd argument (*buffer*) of *ux_host_class_video_transfer_buffer_add* function when using DMA transfer.
- Note the following when using *ux_host_class_video_start* function (USBX HUVC API)
    - Please be sure to confirm return value of the *ux_host_class_video_max_payload_get* function is less than or equal to 1024. If the return value is greater than 1024, start video streaming by referring to the following.

```
channel.ux_host_class_video_parameter_channel_bandwidth_selection = 1
024;
status = ux_host_class_video_ioctl(video, UX_HOST_CLASS_VIDEO_IOCTL_C
HANNEL_START, &channel);
```

- The USBHS module does not support the high-bandwidth isochronous transfer (image below), the maximum packet size for isochronous transfer is 1024 bytes and supports only one transaction per microframe.



| Interface Descriptor | Radix: auto |
|---|---|
| bLength | 9 |
| bDescriptorType | INTERFACE (0x04) |
| bInterfaceNumber | 1 |
| bAlternateSetting | 1 |
| bNumEndpoints | 1 |
| bInterfaceClass | Video (0x0e) |
| bInterfaceSubClass | Video Streaming (0x02) |
| bInterfaceProtocol | Undefined (0x00) |
| iInterface | None (0) |

| Endpoint Descriptor | Radix: auto |
|---|---|
| bLength | 7 |
| bDescriptorType | ENDPOINT (0x05) |
| bEndpointAddress | 1 IN (0b10000001) |
| bmAttributes.TransferType | Isochronous (0b01) |
| bmAttributes.SynchronizationType | Asynchronous (0b01) |
| bmAttributes.UsageType | Data endpoint (0b00) |
| wMaxPacketSize.PacketSize | 1024 |
| wMaxPacketSize.Transactions | Three transactions per microframe if HS (0b10) |
| bInterval | 1 |

Figure 219: High Bandwidth Isochronous Transfer

- This module does not support the MCU(RA4M1 and RA2A1) since the RAM size is small.
- Please specify the correct framework size to the 2nd argument(device_framework_length_high_speed) and the 4th argument(device_framework_length_full_speed) in "ux_device_stack_initialize" function. If incorrect size is specified, USB driver does not work properly.
- The following is the limitation when using OTG.
    - The user can not use High-speed module(USB_IP1).
    - The DMA transfer is not supported.
    - The MSC is not supported.
    - The user can not use USB Hub.
- The followings are the limitation when using USB Hub.
    - The USB driver allocates USB PIPE9 for USB Hub.
    - The user can not connect the Hub device under the Hub down port.
    - For HCDC, the user can not connect more than 2 CDC devices to USB Hub.
    - For HPRN, the user can not connect more than 2 PRN devices to USB Hub.
    - For HHID, the user can not connect more than 3 HID devices to USB Hub.
    - For HMSC, the user can not connect more than 3 MSC devices to USB Hub.

- Please use USB Host DFU tool described in the chapter "USB Device DFU Class" in the following page.

  https://learn.microsoft.com/en-us/azure/rtos/usbx/usbx-device-stack-supplemental-2

- USBX DFU can upgrade the firmware only once after resetting MCU.
- USBX DFU does not support the upload feature.

- ux_device_class_cdc_acm_read() API, throughput/speed may be similar in following cases.

    - Read operation in double buffer mode when compared to single buffer mode.
    - Read operation in continuous transfer mode when compared to non-continuous transfer mode.

  Because of the USBX CDC device stack limits the read transfer size buffer to single packet (ie. wMaxPacketSize as per Full-speed or High-speed USB port). Please refer Azure RTOS USBX documentation for more information.

- ux_host_class_cdc_acm_write() and ux_host_class_cdc_acm_read() API, throughput/speed on high-speed port may be similar for CPU read and write operation in double buffer mode when compared to single buffer mode because of the FSP driver limits the maximum transfer request length size to 512 bytes (UX_FSP_MAX_BULK_PAYLOAD).
- ux_host_class_cdc_acm_write() and ux_host_class_cdc_acm_read() API, throughput/speed may be less for DMA read and write operation when compared to CPU mode because of the FSP driver limits the maximum transfer request length size to 512 bytes(UX_FSP_MAX_BULK_PAYLOAD).
- ux_host_class_cdc_acm_write() and ux_host_class_cdc_acm_read() API, throughput/speed may be less for DMA read and write operation compared to USB FreeRTOS HCDC DMA because of the FSP driver limits the maximum transfer request length size to 512 bytes (UX_FSP_MAX_BULK_PAYLOAD).

# Descriptor

Templates for USBX descriptors can be found in ra/fsp/src/rm_usbx_port folder. Also, please be sure to use your vendor ID.
Change the descriptor.c.template file for each class as follows if High-speed mode is used.

- rm_usbx_pcdc_descriptor.c.template
    - Comment on lines 108 and 243.
    - Delete the "//" on lines 109 and 242.
- rm_usbx_pmsc_descriptor.c.template
    - Comment on lines 78 and 153.
    - Delete the "//" on lines 79 and 152.
- rm_usbx_paud_descriptor.c.template
    - Comment on lines 167 and 485.
    - Delete the "//" on lines 168 and 486.

There are two types of templates for PHID descriptor.
Keyboard templates should be referred to rm_usbx_phid_descriptor_keyboard.c.template.
Mouse templates should be referred to rm_usbx_phid_descriptor_mouse.c.template.

# Examples

### USBX PCDC Example

PCDC loopback example is as follows.

```
#define VALUE_105 (105)
#define VALUE_2 (2)
#define VALUE_103 (103)
#define VALUE_93 (93)
/**********************************************************************************
* Function Name : ux_cdc_device0_instance_activate
* Description : Get instance
```

```
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
}
/***********************************************************************************
 * End of function ux_cdc_device0_instance_activate
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
}
/***********************************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 ***********************************************************************************/
UINT apl_status_change_cb (ULONG status)
{
 switch (status)
```

```
    {
 case UX_DEVICE_ATTACHED:
            g_attach = USB_YES;
 break;
 case UX_DEVICE_REMOVED:
            g_attach = USB_NO;
 break;
 default:
 break;
    }
 return UX_SUCCESS;
}
/**********************************************************************************
 * End of function apl_status_change_cb
 *********************************************************************************/
/**********************************************************************************
 * Function Name : usbx_pcdc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *********************************************************************************/
void usbx_pcdc_sample (void)
{
 fsp_err_t err;
    uint32_t ret;
    uint32_t size;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               VALUE_103,
                               g_device_framework_full_speed,
                               VALUE_93,
                               g_string_framework,
                               VALUE_105,
                               g_language_id_framework,
```

```
                                VALUE_2,
                                apl_status_change_cb);
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate   =
ux_cdc_device0_instance_activate;
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
       ux_cdc_device0_instance_deactivate;
    ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                   _ux_device_class_cdc_acm_entry,
                                   1,
                                   0x00,
                                   (void *) &g_ux_device_class_cdc_acm0_parameter);
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
     {
 if (USB_YES == g_attach)
     {
 while (g_cdc == UX_NULL)
     {
                  ;
     }
                ret = _ux_device_class_cdc_acm_read(g_cdc, g_buf, DATA_LEN,
&g_actual_length);
 if (UX_SUCCESS == ret)
     {
                  size = g_actual_length;
                  _ux_device_class_cdc_acm_write(g_cdc, g_buf, size,
&g_actual_length);
     }
     }
     }
    }
}
```

```
/**************************************************************************
 * End of function usbx_pcdc_sample
 **************************************************************************/
```

### USBX HCDC Example

The main functions of the HCDC loopback example are as follows:

1. Virtual UART control settings are configured by transmitting the class request SET_LINE_CODING to the CDC device.
2. Sends receive (Bulk In transfer) requests to a CDC peripheral device and receives data.
3. Loops received data back to the peripheral by means of Bulk Out transfers.

The main loop performs loopback processing in which data received from a CDC peripheral device is transmitted unaltered back to the peripheral.

```
 #define VALUE_100 (100)
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
 FSP_PARAMETER_NOT_USED(host_class);
 if (UX_FSP_DEVICE_INSERTION == event) /* Check if there is a device insertion. */
    {
       p_cdc_acm = (UX_HOST_CLASS_CDC_ACM *) instance;
 if (UX_HOST_CLASS_CDC_DATA_CLASS !=
            p_cdc_acm->ux_host_class_cdc_acm_interface->ux_interface_descriptor.bInte
rfaceClass)
      {
 if (UX_NULL != p_cdc_acm->ux_host_class_cdc_acm_next_instance)
      {
 /* It seems the DATA class is on the second interface. Or we hope ! */
                p_cdc_acm = p_cdc_acm->ux_host_class_cdc_acm_next_instance;
 /* Check again this interface, if this is not the data interface, we give up. */
 if
(p_cdc_acm->ux_host_class_cdc_acm_interface->ux_interface_descriptor.bInterfaceClass
!=
                   UX_HOST_CLASS_CDC_DATA_CLASS)
      {
```

```
 /* We did not find a proper data interface. */
                    p_cdc_acm = UX_NULL;
      }
      }
      }
 if (p_cdc_acm != UX_NULL)
      {
            tx_event_flags_set(&g_cdcacm_activate_event_flags0, CDCACM_FLAG, TX_OR);
      }
    }
 else if (event == UX_FSP_DEVICE_REMOVAL) /* Check if there is a device removal. */
    {
       tx_event_flags_set(&g_cdcacm_activate_event_flags0, ~CDCACM_FLAG, TX_AND);
       p_cdc_acm = UX_NULL;
    }
 else
    {
 /* None */
    }
 return UX_SUCCESS;
}
void buffer_clear (uint8_t * p)
{
    uint16_t counter;
 for (counter = 0; counter < DATA_LEN; counter++)
    {
       *p = 0U;
    }
}
/*****************************************************************************
 * Function Name : usbx_hcdc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
```

```
 ***********************************************************************/
/* CDCACM Host Thread entry function */
void usbx_hcdc_sample (void)
{
    uint32_t status;
    ULONG    actual_flags;
    uint16_t counter = 0;
 for (counter = 0; counter < DATA_LEN; counter++)
    {
       g_write_buf[counter] = (uint8_t) counter;
    }
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(ux_host_usr_event_notification);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
       tx_event_flags_get(&g_cdcacm_activate_event_flags0, CDCACM_FLAG, TX_OR,
&actual_flags, TX_WAIT_FOREVER);
 if (UX_NULL != p_cdc_acm)
     {
 if (0 == g_is_communicate)
     {
                tx_thread_sleep(VALUE_100);
                g_is_communicate = 1;
     }
 if (UX_NULL != p_cdc_acm)
     {
                status = ux_host_class_cdc_acm_write(p_cdc_acm, g_write_buf,
DATA_LEN, &g_write_actual_length);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_write_actual_length))
     {
                   g_read_actual_length = 0;
                   buffer_clear(g_read_buf);
 if (UX_NULL != p_cdc_acm)
```

```
     {
                          status = ux_host_class_cdc_acm_read(p_cdc_acm, g_read_buf,
DATA_LEN, &g_read_actual_length);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_read_actual_length))
     {
 for (counter = 0; counter < DATA_LEN; counter++)
     {
 if ((uint8_t) counter != g_read_buf[counter])
     {
 while (1)
     {
                    ;
     }
     }
     } /* for */
     }
     } /* UX_NULL != p_cdc_acm */
     }
     } /* UX_NULL != p_cdc_acm */
     }
   }
}
```

### USBX PMSC Example

PMSC storage example is as follows.

```
const rm_block_media_cfg_t g_rm_block_media0_cfg =
{.p_extend = NULL, .p_callback = NULL, .p_context = NULL, };
rm_block_media_info_t g_rm_block_info0 =
{.sector_size_bytes = STRG_MEDIASIZE, .num_sectors = STRG_TOTALSECT, .reentrant =
false, .write_protected = false, };
rm_block_media_instance_t g_rm_block_media0 =
{.p_api = &g_rm_block_media_on_user_media, .p_ctrl = &g_rm_block_info0, .p_cfg =
&g_rm_block_media0_cfg, };
```

```
/***********************************************************************************
 * Function Name : usbx_pmsc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usbx_pmsc_sample (void)
{
 fsp_err_t err;
    UINT      ret;
    ULONG    size;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               60,
                               g_device_framework_full_speed,
                               50,
                               g_string_framework,
                               93,
                               g_language_id_framework,
                               2,
                               UX_NULL);
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
     {
           ;
     }
    }
}
```

### USBX HMSC Example

HMSC storage example is as follows. See usbx_hmsc_sample for the basic operation of HMSC. Also, please refer to usbx_hmsc_sample_format for the format of the USB memory.

```
#define UX_STORAGE_BUFFER_SIZE (64 * 1024)
#define EVENT_USB_PLUG_IN (1UL << 0)
#define EVENT_USB_PLUG_OUT (1UL << 1)
#define MEMPOOL_SIZE (63488)
#define DATA_LEN (2048)
#define VALUE_32 (32)
#define VALUE_100 (100)
#define VALUE_200 (200)
#define VALUE_256 (256)
#define VALUE_1024 (1024)
#define DEVICE_INSERTION (0x01U)
#define DEVICE_REMOVAL (0x02U)
static uint16_t g_read_buf[UX_STORAGE_BUFFER_SIZE];
static uint16_t g_write_buf[UX_STORAGE_BUFFER_SIZE];
static FX_FILE g_file;
static UCHAR g_fx_media0_media_memory[UX_STORAGE_BUFFER_SIZE];
static uint8_t g_ux_pool_memory[MEMPOOL_SIZE];
static FX_MEDIA * g_p_media = UX_NULL;
TX_EVENT_FLAGS_GROUP g_usb_plug_events;
UINT usb_host_plug_event_notification(ULONG usb_event, UX_HOST_CLASS * host_class,
VOID * instance);
UINT ux_system_host_storage_fx_media_get(UX_HOST_CLASS_STORAGE       * instance,
                                         UX_HOST_CLASS_STORAGE_MEDIA **
p_storage_media,
                                         FX_MEDIA                  ** p_fx_media);
static UINT apl_change_function (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
    UINT                         status = UX_SUCCESS;
    UX_HOST_CLASS               * class;
    UX_HOST_CLASS_STORAGE       * storage;
    UX_HOST_CLASS_STORAGE_MEDIA * storage_media;
 FSP_PARAMETER_NOT_USED(host_class);
 FSP_PARAMETER_NOT_USED(instance);
```

```
/* Check if there is a device insertion. */
if (DEVICE_INSERTION == event)
   {
      status = ux_host_stack_class_get(_ux_system_host_class_storage_name, &class);
if (UX_SUCCESS != status)
     {
return status;
     }
      status = ux_host_stack_class_instance_get(class, 0, (void **) &storage);
if (UX_SUCCESS != status)
     {
return status;
     }
if (UX_HOST_CLASS_INSTANCE_LIVE != storage->ux_host_class_storage_state)
     {
return UX_ERROR;
     }
      storage_media = class->ux_host_class_media;
      g_p_media    = &storage_media->ux_host_class_storage_media;
      tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_IN, TX_OR);
   }
else if (DEVICE_REMOVAL == event) /* Check if there is a device removal. */
   {
      g_p_media = UX_NULL;
      tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR);
   }
else
   {
/* None */
   }
return status;
}
/***********************************************************************************
 * Function Name : usbx_hmsc_sample
```

```
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usbx_hmsc_sample (void)
{
    ULONG      actual_length = 0;
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       fx_return;
    uint16_t   data_count = 0;
    FX_MEDIA * p_media    = UX_NULL;
    CHAR       volume[VALUE_32];
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
 // Wait until device inserted.
       tx_return = tx_event_flags_get(&g_usb_plug_events,
                                      EVENT_USB_PLUG_IN,
                                      TX_OR_CLEAR,
                                      &actual_flags,
                                      TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 // Get the pointer to FileX Media Control Block for a USB flash device
       p_media = g_p_media;
 // Retrieve the volume name of the opened media from the Data sector
       fx_return = fx_media_volume_get(p_media, volume, FX_DIRECTORY_SECTOR);
```

```
 if (FX_SUCCESS == fx_return)
      {
 // Set the default directory in the opened media, arbitrary name called "firstdir"
            fx_directory_default_set(p_media, "firstdir");
 // Suspend this thread for 200 time-ticks
            tx_thread_sleep(VALUE_100);
 // Try to open the file, 'counter.txt'.
            fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 // The 'counter.txt' file is not found, so create a new file
               fx_return = fx_file_create(p_media, "counter.txt");
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
 // Open that file
             fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
      }
 // Already open a file, then read the file in blocks
 // Set a specified byte offset for reading
            fx_return = fx_file_seek(&g_file, 0);
 if (FX_SUCCESS == fx_return)
      {
               fx_return = fx_file_read(&g_file, g_read_buf, DATA_LEN,
&actual_length);
 if ((FX_SUCCESS == fx_return) || (FX_END_OF_FILE == fx_return))
      {
```

```
 if (data_count == VALUE_1024)
      {
 // empty file
                        data_count = 0;
      }
 for (uint16_t data_max_count = data_count; data_count < (data_max_count +
VALUE_256); data_count++)
      {
                        g_write_buf[data_count] = data_count;
      }
 // Set the specified byte offset for writing
              fx_return = fx_file_seek(&g_file, 0);
 if (FX_SUCCESS == fx_return)
      {
 // Write the file in blocks
                        fx_return = fx_file_write(&g_file, g_write_buf, DATA_LEN);
 if (FX_SUCCESS == fx_return)
      {
      }
 else
      {
              tx_thread_sleep(TX_WAIT_FOREVER);
      }
      }
      }
      }
 else
      {
              tx_thread_sleep(TX_WAIT_FOREVER);
      }
 // Close already opened file
            fx_return = fx_file_close(&g_file);
 if (FX_SUCCESS != fx_return)
      {
```

```
              tx_thread_sleep(TX_WAIT_FOREVER);
     }
           tx_thread_sleep(VALUE_200);
     }
 else
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 /* flush the media */
      fx_return = fx_media_flush(p_media);
 if (FX_SUCCESS != fx_return)
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 /* close the media */
      fx_return = fx_media_close(p_media);
 if (FX_SUCCESS != fx_return)
     {
           tx_thread_sleep(TX_WAIT_FOREVER);
     }
 // Wait for unplugging the USB
      tx_event_flags_get(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR_CLEAR,
&actual_flags, TX_WAIT_FOREVER);
    } // while(1)
}
void usbx_hmsc_sample_format (void)
{
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       status = UX_SUCCESS;
    FX_MEDIA * p_media = UX_NULL;
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
```

```
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 // Wait until device inserted.
    tx_return = tx_event_flags_get(&g_usb_plug_events, EVENT_USB_PLUG_IN,
TX_OR_CLEAR, &actual_flags, TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
    {
       tx_thread_sleep(TX_WAIT_FOREVER);
    }
 // Get the pointer to FileX Media Control Block for a USB flash device
    p_media = g_p_media;
    memset(g_fx_media0_media_memory, 0x00, sizeof(g_fx_media0_media_memory));
    status = fx_media_format(p_media,                                  // Pointer to
FileX media control block.
                             p_media->fx_media_driver_entry,          // Driver entry
                             p_media->fx_media_driver_info,            // Pointer to
Block Media Driver
                             g_fx_media0_media_memory,                // Media buffer
pointer
                             p_media->fx_media_memory_size,            // Media buffer
size
 "sample",                                 // Volume Name
                             p_media->fx_media_number_of_FATs,         // Number of
FATs
                             p_media->fx_media_root_directory_entries, // Directory
Entries
                             p_media->fx_media_hidden_sectors,         // Hidden
sectors
                             (ULONG) p_media->fx_media_total_sectors, // Total sectors
                             p_media->fx_media_bytes_per_sector,       // Sector size
                             p_media->fx_media_sectors_per_cluster,    // Sectors per
cluster
                             p_media->fx_media_heads,                  // Heads (disk
media)
```

```
                            p_media->fx_media_sectors_per_track);
 if ((uint8_t) FX_SUCCESS != status)
    {
       __BKPT(0);
    }
}
```

### USBX PHID Example

PHID keyboard example is as follows.

```
/**************************************************************************
 * Function Name : usbx_phid_keyboard_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 **************************************************************************/
void usbx_phid_keyboard_sample (void)
{
    UX_SLAVE_CLASS_HID_EVENT hid_event;
    UCHAR    key;
 fsp_err_t err;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_device_stack_initialize(NULL,
                               0,
                               g_device_framework_full_speed,
#if defined(APL_OUT_TRANSFER)
                               VALUE_59,
#else /* defined(APL_OUT_TRANSFER) */
                               VALUE_52,
#endif /* defined(APL_OUT_TRANSFER) */
                               g_string_framework,
                               VALUE_53,
                               g_language_id_framework,
                               VALUE_2,
```

```
                                apl_status_change_cb);
    g_ux_device_class_hid_parameter.ux_slave_class_hid_instance_activate         =
ux_hid_instance_activate;
    g_ux_device_class_hid_parameter.ux_slave_class_hid_instance_deactivate       =
ux_hid_instance_deactivate;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_address =
g_apl_report;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_length =
REPORT_DESCRIPTOR_LENGTH;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_callback       =
apl_hid_set_report_cb;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_id      = 0;
#if defined(APL_OUT_TRANSFER)
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_receiver_initialize
=
       ux_device_class_hid_receiver_initialize;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_receiver_event_max_
number = 16;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_receiver_event_max_
length = DATA_LEN;
#endif  /* defined(APL_OUT_TRANSFER) */
    ux_device_stack_class_register(_ux_system_slave_class_hid_name,
                                   _ux_device_class_hid_entry,
                                   1,
                                   0x00,
                                   (void *) &g_ux_device_class_hid_parameter);
 /* Set the first key to 'a' which is 04. */
    key = 0x04;
 /* reset the HID event structure. */
    ux_utility_memory_set(&hid_event, 0, sizeof(UX_SLAVE_CLASS_HID_EVENT));
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
```

```
    {
if (USB_NO == g_suspend)
    {
while (UX_NULL == g_hid)
    {
/* Then wait. */
                tx_thread_sleep(10);
    }
/* 5sec wait */
              usb_cpu_delay_xms((uint16_t) VALUE_5000);
/* Then insert a key into the keyboard event. Length is fixed to 8. */
              hid_event.ux_device_class_hid_event_length = 8;
/* First byte is a modifier byte. */
              hid_event.ux_device_class_hid_event_buffer[0] = 0;
/* Second byte is reserved. */
              hid_event.ux_device_class_hid_event_buffer[1] = 0;
/* The 6 next bytes are keys. We only have one key here. */
              hid_event.ux_device_class_hid_event_buffer[2] = key;
if (UX_NULL != g_hid)
    {
/* Set the keyboard event. */
                ux_device_class_hid_event_set(g_hid, &hid_event);
    }
/* Next event has the key depressed. */
              hid_event.ux_device_class_hid_event_buffer[2] = 0;
/* Length is fixed to 8. */
              hid_event.ux_device_class_hid_event_length = 8;
if (UX_NULL != g_hid)
    {
/* Set the keyboard event. */
                ux_device_class_hid_event_set(g_hid, &hid_event);
/* Are we at the end of alphabet ? */
if (key != (0x04 + 26))
    {
```

```
 /* Next key. */
                        key++;
     }
 else
     {
 /* Start over again. */
     key = 0x04;
     }
     }
#if defined(APL_OUT_TRANSFER)
 if (UX_NULL != g_hid)
     {
                 status = ux_device_class_hid_receiver_event_get(g_hid,
&hid_out_event);
 if (UX_SUCCESS == status)
     {
 for (i = 0; i < hid_out_event.ux_device_class_hid_received_event_length; i++)
     {
                           g_out_buf[i] =
*(hid_out_event.ux_device_class_hid_received_event_data + i);
     }
                     ux_device_class_hid_receiver_event_free(g_hid);
     }
     }
#endif  /* defined(APL_OUT_TRANSFER) */
     }
 else
     {
 if (USB_NO == g_remote_wakeup)
     {
                 tx_thread_sleep(VALUE_10000);
 /* Remote wakeup processing */
                 g_remote_wakeup = USB_YES;
                 ux_device_stack_host_wakeup();
```

```
      }
      }
      }
    }
}
```

### USBX HHID Example

HHID user interface example is as follows.

```
void keyboard_update_task (ULONG thread_input)
{
    ULONG usbx_return_value;
 /* keyboard button masks, set by ux_host_class_hid_keyboard_buttons_get call */
    ULONG keyboard_key = 0;
 /* keyboard state masks, set by ux_host_class_hid_keyboard_buttons_get call */
    ULONG keyboard_state = 0;
#if defined(UX_HOST_CLASS_HID_INTERRUPT_OUT_SUPPORT)
    UX_HOST_CLASS_HID_KEYBOARD * keyboard_instance;
    ULONG i;
    UX_HOST_CLASS_HID_CLIENT_REPORT client_report;
    UX_HOST_CLASS_HID_REPORT       hid_report;
 for (i = 0; i < (OUT_DATA_LEN / 4); i++)
    {
       g_buf[i] = i;
    }
#endif  /* defined(UX_HOST_CLASS_HID_INTERRUPT_OUT_SUPPORT) */
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
 if (UX_NULL != hid_keyboard_client)
      {
            usbx_return_value = ux_host_class_hid_keyboard_key_get(
                (UX_HOST_CLASS_HID_KEYBOARD *)
hid_keyboard_client->ux_host_class_hid_client_local_instance,
```

```
                &keyboard_key,
                &keyboard_state);
 if ((usbx_return_value == UX_SUCCESS) || (usbx_return_value == UX_NO_KEY_PRESS))
     {
                hid_devices_info.device_connected = KEYBOARD_DEVICE;
                hid_devices_info.key             = keyboard_key;
                hid_devices_info.keyboard_status = keyboard_state;
 /* Clear the states for next read */
                keyboard_key   = 0;
                keyboard_state = 0;
 /* copy the keyboard states to queue */
                tx_queue_send(&device_parameters, &hid_devices_info, TX_NO_WAIT);
#if defined(UX_HOST_CLASS_HID_INTERRUPT_OUT_SUPPORT)
                hid_report.ux_host_class_hid_report_id          = 0;
                hid_report.ux_host_class_hid_report_type       =
UX_HOST_CLASS_HID_REPORT_TYPE_OUTPUT;
                hid_report.ux_host_class_hid_report_byte_length = OUT_DATA_LEN;
                client_report.ux_host_class_hid_client_report       = &hid_report;
                client_report.ux_host_class_hid_client_report_buffer = g_buf;
                client_report.ux_host_class_hid_client_report_length = OUT_DATA_LEN;
                client_report.ux_host_class_hid_client_report_flags =
UX_HOST_CLASS_HID_REPORT_RAW;
                keyboard_instance =
                   (UX_HOST_CLASS_HID_KEYBOARD *)
hid_keyboard_client->ux_host_class_hid_client_local_instance;
 /* HID Out Transfer */
                ux_host_class_hid_report_set((UX_HOST_CLASS_HID *)
keyboard_instance->ux_host_class_hid_keyboard_hid,
                                             &client_report);
#endif  /* defined(UX_HOST_CLASS_HID_INTERRUPT_OUT_SUPPORT) */
     }
     }
       tx_thread_sleep(10);
    }
```

```
}
void mouse_update_task (ULONG thread_input)
{
 /* mouse button masks, set by ux_host_class_hid_mouse_buttons_get call */
    ULONG mouse_buttons;
 /* X co-ordinate displacement of mouse */
    SLONG mouse_x_position = 0;
 /* Y co-ordinate displacement of mouse */
    SLONG mouse_y_position = 0;
 /* variable to hold USBX return values */
    ULONG usbx_return_value;
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
 if (UX_SUCCESS != hid_mouse_client)
     {
           usbx_return_value = ux_host_class_hid_mouse_buttons_get(
               (UX_HOST_CLASS_HID_MOUSE *)
hid_mouse_client->ux_host_class_hid_client_local_instance,
               &mouse_buttons);
 if (UX_SUCCESS == usbx_return_value)
     {
               usbx_return_value = ux_host_class_hid_mouse_position_get(
                 (UX_HOST_CLASS_HID_MOUSE *)
hid_mouse_client->ux_host_class_hid_client_local_instance,
                  &mouse_x_position,
                  &mouse_y_position);
     }
 if (UX_SUCCESS == usbx_return_value)
     {
               hid_devices_info.device_connected = MOUSE_DEVICE;
               hid_devices_info.key               = mouse_buttons;
               hid_devices_info.mouse_direction_X = mouse_x_position;
               hid_devices_info.mouse_direction_Y = mouse_y_position;
```

```
                tx_queue_send(&device_parameters, &hid_devices_info, TX_NO_WAIT);
      }
      }
       tx_thread_sleep(10);
    }
}
void usbx_hhid_sample (void)
{
    uint8_t i;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(ux_system_host_hid_change_function);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    tx_thread_create(&keyboard_update,
                     (CHAR *) "keyboard_update_task",
                     keyboard_update_task,
                     (ULONG) NULL,
                     &keyboard_update_stack,
                     2048,
                     22,
                     2,
                     1,
                     TX_AUTO_START);
    tx_thread_create(&mouse_update,
                     (CHAR *) "mouse_update_task",
                     mouse_update_task,
                     (ULONG) NULL,
                     &mouse_update_stack,
                     1024,
                     22,
                     2,
                     1,
                     TX_AUTO_START);
 while (1)
    {
```

```
 for (i = 0; i < UX_HOST_CLASS_HID_MAX_CLIENTS; i++)
      {
 /* Check whether the instance registered? through USB HID device insertion callback?
*/
 if (UX_NULL != hid_class_keyboard_instance[i])
      {
                UX_HOST_CLASS_HID_CLIENT * hid_client =
hid_class_keyboard_instance[i]->ux_host_class_hid_client;
                hid_keyboard_client = hid_client;
         tx_thread_sleep(10);
      }
 /* Check whether the instance registered? through USB HID device insertion callback?
*/
 if (UX_NULL != hid_class_mouse_instance[i])
      {
                UX_HOST_CLASS_HID_CLIENT * hid_client =
hid_class_mouse_instance[i]->ux_host_class_hid_client;
                hid_mouse_client = hid_client;
         tx_thread_sleep(10);
      }
 /* If multiple similar type devices are connected, allow them one by one share data
with application */
         tx_thread_sleep(10);
      }
    }
}
```

### USBX PAUD Example

PAUD example is as follows.

```
#define VALUE_275 (275)
#define VALUE_226 (226)
#define VALUE_93 (93)
#define VALUE_2 (2)
```

```
#define STACK_SIZE (1024U)
#define NUM_OF_FRAME (8U)
#define USB_MAX_PACKET_SIZE_IN (200U)
#define USB_MAX_PACKET_SZIE_OUT (192U)
#define USB_APL_ON (1U)
#define USB_APL_OFF (0U)
/*** Please enable the following macro when supporting High-speed. ***/
// #define APL_AUDIO_20
/***********************************************************************/
static uint8_t g_read_buf[NUM_OF_FRAME][USB_MAX_PACKET_SZIE_OUT];
static uint8_t g_write_buf[USB_MAX_PACKET_SIZE_IN];
static UX_DEVICE_CLASS_AUDIO * volatile g_p_audio = UX_NULL;
static volatile uint8_t g_read_wp = 0U;
static volatile uint32_t g_read_alternate_setting = USB_APL_OFF;
static volatile uint32_t g_write_alternate_setting = USB_APL_OFF;
static volatile uint8_t g_apl_usb_status = USB_APL_DETACH;
#ifdef APL_AUDIO_20
static UX_DEVICE_CLASS_AUDIO20_CONTROL_GROUP g_audio20_control_group;
static UX_DEVICE_CLASS_AUDIO20_CONTROL g_audio20_control[2];
#else  /* APL_AUDIO_20 */
static UX_DEVICE_CLASS_AUDIO10_CONTROL_GROUP g_audio_control_group;
static UX_DEVICE_CLASS_AUDIO10_CONTROL g_audio_control[2];
#endif  /* APL_AUDIO_20 */
/**************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 **************************************************************************/
static UINT apl_status_change_cb (ULONG status)
{
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
```

```
     {
           g_apl_usb_status = USB_APL_DEFAULT;
break;
     }
case UX_DEVICE_CONFIGURED:
     {
           g_apl_usb_status = USB_APL_CONFIGURED;
break;
     }
case UX_DEVICE_REMOVED:
     {
           g_apl_usb_status = USB_APL_DETACH;
           g_read_wp                = 0U;
           g_read_alternate_setting = USB_APL_OFF;
           g_write_alternate_setting = USB_APL_OFF;
break;
     }
case UX_DEVICE_SUSPENDED:
     {
if (USB_APL_CONFIGURED == g_apl_usb_status)
     {
              g_apl_usb_status = USB_APL_SUSPEND;
     }
break;
     }
case UX_DEVICE_RESUMED:
     {
if (USB_APL_SUSPEND == g_apl_usb_status)
     {
             g_apl_usb_status = USB_APL_CONFIGURED;
     }
break;
     }
default:
```

```
      {
 break;
      }
    }
 return UX_SUCCESS;
}
/**********************************************************************************
 * End of function apl_status_change_cb
 *********************************************************************************/
/**********************************************************************************
 * Function Name : apl_audio_read_instance_activate
 * Description : Get instance
 * Arguments : void * p_instance : Pointer to the area store the instance pointer
 * Return value : none
 *********************************************************************************/
static void apl_audio_instance_activate (void * p_instance)
{
 /* Save the CDC instance. */
    g_p_audio = (UX_DEVICE_CLASS_AUDIO *) p_instance;
}
/**********************************************************************************
 * End of function apl_audio_read_instance_activate
 *********************************************************************************/
/**********************************************************************************
 * Function Name : apl_audio_read_instance_deactivate
 * Description : Clear instance
 * Arguments : void * p_instance : Pointer to area store the instance pointer
 * Return value : none
 *********************************************************************************/
static void apl_audio_instance_deactivate (void * p_instance)
{
 FSP_PARAMETER_NOT_USED(p_instance);
    g_p_audio = UX_NULL;
}
```

```
/**************************************************************************
 * End of function apl_audio_read_instance_deactivate
 **************************************************************************/
#ifdef APL_AUDIO_20
/**************************************************************************
 * Function Name : apl_audio20_request_process
 * Description : Audio20 Control Request Processing
 * Arguments : UX_DEVICE_CLASS_AUDIO * : Pointer to Audio instance
 * : UX_SLAVE_TRANSFER * : Pointer to UX_SLAVE_TRANSFER structure
 * Return value : UX_SUCCESS
 **************************************************************************/
static UINT apl_audio20_request_process (UX_DEVICE_CLASS_AUDIO * p_audio,
UX_SLAVE_TRANSFER * p_transfer)
{
    UINT    ux_err;
    uint8_t i;
    uint8_t number;
    ux_err = ux_device_class_audio20_control_process(p_audio, p_transfer,
&g_audio20_control_group);
 if (UX_SUCCESS == ux_err)
    {
 /* Request handled, check changes */
       number = (uint8_t)
g_audio20_control_group.ux_device_class_audio20_control_group_controls_nb;
       for (i = 0; i < number; i++)
     {
 switch (g_audio20_control[i].ux_device_class_audio20_control_changed)
     {
 case UX_DEVICE_CLASS_AUDIO20_CONTROL_MUTE_CHANGED:
     {
                  g_control_mute[i] =
g_audio20_control[i].ux_device_class_audio20_control_mute[0];
 break;
     }
```

```
 case UX_DEVICE_CLASS_AUDIO20_CONTROL_VOLUME_CHANGED:
      {
                    g_control_volume[i] =
g_audio20_control[i].ux_device_class_audio20_control_volume[0];
 break;
      }
 default:
      {
 break;
      }
      }
      }
    }
 return UX_SUCCESS;
}
/**********************************************************************************
 * End of function apl_audio20_request_process
 *********************************************************************************/
#else  /* APL_AUDIO_20 */
/**********************************************************************************
 * Function Name : apl_audio10_request_process
 * Description : Audio10 Control Request Processing
 * Arguments : UX_DEVICE_CLASS_AUDIO * : Pointer to Audio instance
 * : UX_SLAVE_TRANSFER * : Pointer to UX_SLAVE_TRANSFER structure
 * Return value : UX_SUCCESS
 *********************************************************************************/
static UINT apl_audio10_request_process (UX_DEVICE_CLASS_AUDIO * p_audio,
UX_SLAVE_TRANSFER * p_transfer)
{
    UINT    ux_err;
    uint8_t i;
    uint8_t number;
    ux_err = ux_device_class_audio10_control_process(p_audio, p_transfer,
&g_audio_control_group);
```

```
 if (UX_SUCCESS == ux_err)
    {
 /* Request handled, check changes */
       number = (uint8_t)
g_audio_control_group.ux_device_class_audio10_control_group_controls_nb;
       for (i = 0; i < number; i++)
      {
 switch (g_audio_control[i].ux_device_class_audio10_control_changed)
      {
 case UX_DEVICE_CLASS_AUDIO10_CONTROL_MUTE_CHANGED:
      {
                  g_control_mute[i] =
g_audio_control[i].ux_device_class_audio10_control_mute[0];
 break;
      }
 case UX_DEVICE_CLASS_AUDIO10_CONTROL_VOLUME_CHANGED:
      {
                  g_control_volume[i] =
g_audio_control[i].ux_device_class_audio10_control_volume[0];;
 break;
      }
 default:
      {
 break;
      }
      }
      }
    }
 return UX_SUCCESS;
}
/**********************************************************************************
 * End of function apl_audio10_request_process
 *********************************************************************************/
#endif  /* APL_AUDIO_20 */
```

```
/**************************************************************************
 * Function Name : apl_audio_read_change
 * Description : Callback function called when switching alternate setting value of
OUT transfer
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Alternate Setting Value
 * Return value : UX_SUCCESS
 **************************************************************************/
static void apl_audio_read_change (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
alternate_setting)
{
    UINT ux_err;
 if (USB_APL_ON == alternate_setting)
    {
       ux_device_class_audio_reception_start(p_stream);
    }
 else
    {
 if (USB_APL_ON == g_read_alternate_setting)
      {
 /* Alternate Setting 1 --> 0 */
            g_read_wp = 0U;
      }
    }
    g_read_alternate_setting = alternate_setting;
}
/**************************************************************************
 * End of function apl_audio_read_change
 **************************************************************************/
/**************************************************************************
 * Function Name : apl_audio_read_done
 * Description : Callback function called when completing of OUT transfer reception
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
```

```
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Actual Length
 * Return value : UX_SUCCESS
 ***********************************************************************/
static void apl_audio_read_done (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
actual_length)
{
    UINT    ux_err;
    UCHAR * p_buffer;
    ULONG   length;
    UINT    i;
 FSP_PARAMETER_NOT_USED(actual_length);
 if (USB_APL_ON == g_read_alternate_setting)
    {
       ux_err = ux_device_class_audio_read_frame_get(p_stream, &p_buffer, &length);
 if (UX_SUCCESS == ux_err)
      {
 for (i = 0; i < length; i++)
      {
                g_read_buf[g_read_wp][i] = *(p_buffer + i);
      }
            g_read_wp++;
            g_read_wp %= NUM_OF_FRAME;
            ux_device_class_audio_read_frame_free(p_stream);
      }
    }
}
/***********************************************************************
 * End of function apl_audio_read_done
 ***********************************************************************/
/***********************************************************************
 * Function Name : apl_audio_write_change
 * Description : Callback function called when switching alternate setting value of
IN transfer
```

```
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Alternate Setting Value
 * Return value : UX_SUCCESS
 ***********************************************************************/
static void apl_audio_write_change (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
alternate_setting)
{
    UINT ux_err;
 if (USB_APL_ON == alternate_setting)
    {
       ux_err = ux_device_class_audio_frame_write(p_stream, g_write_buf,
USB_MAX_PACKET_SIZE_IN);
 if (UX_SUCCESS == ux_err)
      {
            ux_device_class_audio_transmission_start(p_stream);
      }
    }
    g_write_alternate_setting = alternate_setting;
}
/***********************************************************************
 * End of function apl_audio_write_change
 ***********************************************************************/
/***********************************************************************
 * Function Name : apl_audio_write_done
 * Description : Callback function called when completing of IN transfer transmission
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Actual Length
 * Return value : None
 ***********************************************************************/
static void apl_audio_write_done (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
actual_length)
{
```

```
 FSP_PARAMETER_NOT_USED(actual_length);
 if (USB_APL_ON == g_write_alternate_setting)
    {
       ux_device_class_audio_frame_write(p_stream, g_write_buf,
USB_MAX_PACKET_SIZE_IN);
    }
}
/******************************************************************************
 * End of function apl_audio_write_done
 ******************************************************************************/
/******************************************************************************
 * Function Name : usbx_paudio_apl_init
 * Description : Initialization processing
 * Arguments : None
 * Return value : None
 ******************************************************************************/
void usbx_paudio_apl_init (void)
{
 fsp_err_t err;
    UINT      ux_err;
    uint16_t i = 0;
    UX_DEVICE_CLASS_AUDIO_STREAM_PARAMETER audio_stream_parameter[2];
    UX_DEVICE_CLASS_AUDIO_PARAMETER       audio_parameter;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               VALUE_275,
                               g_device_framework_full_speed,
                               VALUE_226,
                               g_string_framework,
                               VALUE_93,
                               g_language_id_framework,
                               VALUE_2,
                               apl_status_change_cb);
 /* Read Initialization */
```

```
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_change =
       apl_audio_read_change;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_frame_done =
       apl_audio_read_done;

audio_stream_parameter[0].ux_device_class_audio_stream_parameter_thread_stack_size
= STACK_SIZE;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_max_frame_buffer
_nb   = NUM_OF_FRAME;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_max_frame_buffer
_size = USB_MAX_PACKET_SZIE_OUT;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_thread_entry
      =
       ux_device_class_audio_read_thread_entry;
 /* Write Initialization */
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_change =
       apl_audio_write_change;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_frame_done =
       apl_audio_write_done;

audio_stream_parameter[1].ux_device_class_audio_stream_parameter_thread_stack_size
= STACK_SIZE;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_max_frame_buffer
_nb   = NUM_OF_FRAME;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_max_frame_buffer
_size = USB_MAX_PACKET_SIZE_IN;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_thread_entry
      =
       ux_device_class_audio_write_thread_entry;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_slave_class_audio_in
```

```
stance_activate =
       apl_audio_instance_activate;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_slave_class_audio_in
stance_deactivate =
       apl_audio_instance_deactivate;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_device_class_audio_c
ontrol_process
#ifdef APL_AUDIO_20
       = apl_audio20_request_process;
#else
       = apl_audio10_request_process;
#endif
    audio_parameter.ux_device_class_audio_parameter_streams_nb = 2;
    audio_parameter.ux_device_class_audio_parameter_streams    =
&audio_stream_parameter[0];
    ux_err =
       ux_device_stack_class_register(_ux_system_slave_class_audio_name,
                                      _ux_device_class_audio_entry,
                                   1,
                                   0x00,
                                      (void *) &audio_parameter);
 if (UX_SUCCESS != ux_err)
    {
       USB_APL_AUDIO_ERROR();
    }
#ifdef APL_AUDIO_20
    g_audio20_control[0].ux_device_class_audio20_control_cs_id             = 0x10;
    g_audio20_control[0].ux_device_class_audio20_control_sampling_frequency = 48000;
    g_audio20_control[0].ux_device_class_audio20_control_fu_id             = 5;
    g_audio20_control[0].ux_device_class_audio20_control_mute[0]            = 0;
    g_audio20_control[0].ux_device_class_audio20_control_volume_min[0]      = 0;
    g_audio20_control[0].ux_device_class_audio20_control_volume_max[0]      = 100;
    g_audio20_control[0].ux_device_class_audio20_control_volume[0]          = 50;
    g_audio20_control[1].ux_device_class_audio20_control_cs_id             = 0x10;
```

```
    g_audio20_control[1].ux_device_class_audio20_control_sampling_frequency = 48000;
    g_audio20_control[1].ux_device_class_audio20_control_fu_id             = 8;
    g_audio20_control[1].ux_device_class_audio20_control_mute[0]            = 0;
    g_audio20_control[1].ux_device_class_audio20_control_volume_min[0]      = 0;
    g_audio20_control[1].ux_device_class_audio20_control_volume_max[0]      = 200;
    g_audio20_control[1].ux_device_class_audio20_control_volume[0]          = 100;
    g_audio20_control_group.ux_device_class_audio20_control_group_controls_nb = 2;
    g_audio20_control_group.ux_device_class_audio20_control_group_controls    =
&g_audio20_control[0];
#else  /* APL_AUDIO_20 */
    g_audio_control[0].ux_device_class_audio10_control_fu_id         = 5;
    g_audio_control[0].ux_device_class_audio10_control_mute[0]       = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume[0]    = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume_min[0] = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume_max[0] = 0x80;
    g_audio_control[0].ux_device_class_audio10_control_volume_res[0] = 0x40;
    g_audio_control[1].ux_device_class_audio10_control_fu_id                = 8;
    g_audio_control[1].ux_device_class_audio10_control_mute[0]             = 0x10;
    g_audio_control[1].ux_device_class_audio10_control_volume[0]            = 0x00;
    g_audio_control[1].ux_device_class_audio10_control_volume_min[0]       = 0;
    g_audio_control[1].ux_device_class_audio10_control_volume_max[0]       = 0xF0;
    g_audio_control[1].ux_device_class_audio10_control_volume_res[0]       = 0x80;
    g_audio_control_group.ux_device_class_audio10_control_group_controls_nb = 2;
    g_audio_control_group.ux_device_class_audio10_control_group_controls    =
&g_audio_control[0];
#endif  /* APL_AUDIO_20 */
 for (i = 0; i < USB_MAX_PACKET_SIZE_IN; i++)
    {
       g_write_buf[i] = (UCHAR) (i & 0xFF);
    }
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
       USB_APL_AUDIO_ERROR();
```

```
    }
}
/***********************************************************************************
 * End of function usbx_paudio_apl_init
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : usbx_paudio_apl
 * Description : Application task for USB Audio
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usbx_paud_sample (void)
{
    usbx_paudio_apl_init();
 while (1)
    {
 switch (g_apl_usb_status)
     {
 case USB_APL_CONFIGURED:
     {
 /* Application Processing */
 break;
     }
 case USB_APL_DETACH:
     {
 break;
     }
 case USB_APL_SUSPEND:
     {
 break;
     }
 default:
     {
 break;
```

```
      }
      }
    }
}
```

### USBX PPRN Example

PPRN example is as follows.

Note: Define "2" to "DEMO_PROTOCOL" macro when supporting IN transfer and define "1" to "DEMO_PROTOCOL" macro when not supporting IN transfer.

```
/*****************************************************************************
 * Macro definitions
 *****************************************************************************/
#define DEMO_PROTOCOL (1U) /* 1-Uni-dir, 2-Bi-dir */
#define DEMO_STACK_SIZE 1024
/* USBx device configuration settings */
#define DEVICE_FRAME_LENGTH_HIGH_SPEED (53U) + ((DEMO_PROTOCOL > 1) ? 7 : 0) /*
Length of g_device_framework_hi_speed[] */
#define DEVICE_FRAME_LENGTH_FULL_SPEED (43U) + ((DEMO_PROTOCOL > 1) ? 7 : 0) /*
Length of g_device_framework_full_speed[] */
#define STRING_FRAMEWORK_LENGTH (53U)  /* Length of g_string_framework[]. If edit
g_string_framework[], need to change this value. */
#define LANGUAGE_ID_FRAME_WORK_LENGTH (2U)
#define CONFIG_NUMB (1U)
#define INTERFACE_NUMB0 (0x00)
#define INTERFACE_NUMB1 (0x01)
#define MEMPOOL_SIZE (18432U)
#define BYTE_SIZE (4U)
#define DATA_LEN (512U)
#define MAX_PACKET_SIZE_HS (512U)
#define MAX_PACKET_SIZE_FS (64U)
#define PRINTER_DEVICE_ID_LENGTH (91U) /* Length of printer_device_id[]. If edit
printer_device_id[], need to change this value. */
/*****************************************************************************
```

```
 * Exported global variables and functions (to be accessed by other files)
 ***********************************************************************************/
extern uint8_t g_device_framework_full_speed[];
extern uint8_t g_device_framework_hi_speed[];
extern uint8_t g_string_framework[];
extern uint8_t g_language_id_framework[];
extern uint8_t printer_device_id[];
static union _PRINTER_PORT_STATUS
{
 struct
    {
       uint8_t reserved    : 3;
       uint8_t not_error   : 1;
       uint8_t select      : 1;
       uint8_t paper_empty : 1;
    } bm;
    uint8_t value;
} device_printer_port_status;
static struct TEST_DATA_STRUCT
{
    uint32_t mem_usage_max;
    uint32_t mem_usage;
} test_data = {0, 0};
/***********************************************************************************
 * Global functions and variables
 ***********************************************************************************/
extern uint32_t usb_peri_usbx_initialize(uint32_t dcd_io);
/***********************************************************************************
 * Private global variables and functions
 ***********************************************************************************/
static void ux_printer_instance_activate(void * printer_instance);
static void ux_printer_instance_deactivate(void * printer_instance);
static void ux_printer_soft_reset(void * printer_instance);
/* Mempool size of 18k is required for USBX device class pre built libraries
```

```
 * and it is valid only if it with default USBX configurations. */
static uint32_t g_ux_pool_memory[(MEMPOOL_SIZE + DEMO_STACK_SIZE * 3) / BYTE_SIZE];
static UX_DEVICE_CLASS_PRINTER_PARAMETER device_printer_parameter;
static UX_DEVICE_CLASS_PRINTER * device_printer = UX_NULL;
static uint8_t device_printer_buffer[DATA_LEN];
uint8_t _ux_system_slave_class_prn_name[] = "ux_device_class_printer";
/* Define local function prototypes. */
void demo_thread_entry(uint32_t thread_input);
void printer_read_thread_entry(uint32_t thread_input);
void printer_write_thread_entry(uint32_t thread_input);
/* Define global data structures. */
static TX_THREAD demo_thread;
static TX_THREAD printer_read_thread;
#if DEMO_PROTOCOL > 1
static TX_THREAD printer_write_thread;
#endif
/**************************************************************************
 * Function Name : ux_printer_instance_activate
 * Description : Get instance
 * Arguments : void * printer_instance : Pointer to the area store the instance
pointer
 * Return value : none
 **************************************************************************/
static void ux_printer_instance_activate (void * printer_instance)
{
 if (device_printer == UX_NULL)
    {
       device_printer = (UX_DEVICE_CLASS_PRINTER *) printer_instance;
       ux_device_class_printer_ioctl(device_printer,
                                     UX_DEVICE_CLASS_PRINTER_IOCTL_PORT_STATUS_SET,
                                     (void *) device_printer_port_status.value);
    }
}
/**************************************************************************
```

```
 * End of function ux_printer_instance_activate
 **********************************************************************************/
/**********************************************************************************
 * Function Name : ux_printer_instance_deactivate
 * Description : Clear instance
 * Arguments : void * printer_instance : Pointer to the area store the instance
pointer
 * Return value : none
 **********************************************************************************/
static void ux_printer_instance_deactivate (void * printer_instance)
{
 if ((void *) device_printer == printer_instance)
    {
       device_printer = UX_NULL;
    }
}
/**********************************************************************************
 * End of function ux_printer_instance_deactivate
 **********************************************************************************/
/**********************************************************************************
 * Function Name : ux_printer_soft_reset
 * Description : This function does nothing in particular.
 * Arguments : void * printer_instance : Pointer to the area store the instance
pointer
 * Return value : none
 **********************************************************************************/
static void ux_printer_soft_reset (void * printer_instance)
{
}
/**********************************************************************************
 * End of function ux_printer_soft_reset
 **********************************************************************************/
/**********************************************************************************
 * Function Name : usbx_pprn_sample
```

```
 * Description : Initialization for Peripheral Printer
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usbx_pprn_sample (void)
{
/* To check ux api return status */
    UINT status = UX_SUCCESS;
/* To check fsp api return status */
 fsp_err_t err = FSP_SUCCESS;
    uint8_t * stack_pointer;
    uint8_t * memory_pointer;
 /* ux_system_initialization */
    stack_pointer = (uint8_t *) g_ux_pool_memory;
    memory_pointer = stack_pointer + DEMO_STACK_SIZE * 3;
    ux_system_initialize(memory_pointer, MEMPOOL_SIZE, UX_NULL, 0x00);
 /* ux_device stack initialization */
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               DEVICE_FRAME_LENGTH_HIGH_SPEED,
                               g_device_framework_full_speed,
                               DEVICE_FRAME_LENGTH_FULL_SPEED,
                               g_string_framework,
                               STRING_FRAMEWORK_LENGTH,
                               g_language_id_framework,
                               LANGUAGE_ID_FRAME_WORK_LENGTH,
                               UX_NULL);
 /* Set the parameters for callback when insertion/extraction of a printer device. */
    _ux_utility_memory_set(&device_printer_parameter, 0, sizeof
(device_printer_parameter));
    _ux_utility_short_put_big_endian(printer_device_id, PRINTER_DEVICE_ID_LENGTH);
    device_printer_port_status.value = UX_DEVICE_CLASS_PRINTER_SELECT |
UX_DEVICE_CLASS_PRINTER_NOT_ERROR;
    device_printer_parameter.ux_device_class_printer_device_id          =
printer_device_id;
```

```
    device_printer_parameter.ux_device_class_printer_instance_activate   =
ux_printer_instance_activate;
    device_printer_parameter.ux_device_class_printer_instance_deactivate =
ux_printer_instance_deactivate;
    device_printer_parameter.ux_device_class_printer_soft_reset          =
ux_printer_soft_reset;
 /* ux_device stack class registration */
    ux_device_stack_class_register(_ux_system_slave_class_prn_name,
                                   _ux_device_class_printer_entry,
                                   CONFIG_NUMB,
                                   INTERFACE_NUMB0,
                                   (void *) &device_printer_parameter);
 /* Open usb driver */
 R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Create the main demo thread. */
    tx_thread_create(&demo_thread,
 "tx demo",
                     demo_thread_entry,
                     0,
                     stack_pointer,
                     DEMO_STACK_SIZE,
                     20,
                     20,
                     1,
                     TX_AUTO_START);
    stack_pointer += DEMO_STACK_SIZE;
 /* Create the printer read thread. */
    tx_thread_create(&printer_read_thread,
 "read_thread",
                     printer_read_thread_entry,
                     0,
                     stack_pointer,
                     DEMO_STACK_SIZE,
                     20,
```

```
                    20,
                    1,
                    TX_AUTO_START);
    stack_pointer += DEMO_STACK_SIZE;
#if DEMO_PROTOCOL > 1
 /* Create the main demo thread. */
    tx_thread_create(&printer_write_thread,
 "write_thread",
                    printer_write_thread_entry,
                    0,
                    stack_pointer,
                    DEMO_STACK_SIZE,
                    20,
                    20,
                    1,
                    TX_AUTO_START);
#endif
}
/**************************************************************************
 * End of function usbx_pprn_sample
 **************************************************************************/
/**************************************************************************
 * Function Name : apl_mem_usage_update
 * Description : Update memory usage
 * Arguments : none
 * Return value : none
 **************************************************************************/
static void apl_mem_usage_update (void)
{
    uint32_t mem_total;
 /* Update memory usage. */
    mem_total = _ux_system->ux_system_regular_memory_pool_size + (uint32_t) (
       (uint8_t *) _ux_system->ux_system_regular_memory_pool_start -
       (uint8_t *) _ux_system);
```

```
    test_data.mem_usage = mem_total - _ux_system->ux_system_regular_memory_pool_free;
#ifdef UX_ENABLE_MEMORY_STATISTICS
    test_data.mem_usage_max = mem_total -
_ux_system->ux_system_regular_memory_pool_min_free;
#else  /* Not accurate, there could be alloc/free between checks. */
 if (test_data.mem_usage > test_data.mem_usage_max)
    {
       test_data.mem_usage_max = test_data.mem_usage;
    }
#endif
}
/**************************************************************************
 * End of function apl_mem_usage_update
 **************************************************************************/
/**************************************************************************
 * Function Name : apl_minus
 * Description : Adjust timer
 * Arguments : uint32_t v_origin
 * : uint32_t v_minus
 * : uint32_t wrap
 * Return value : Adjusted Tick Value
 **************************************************************************/
static uint32_t apl_minus (uint32_t v_origin, uint32_t v_minus, uint32_t wrap)
{
 if (v_origin >= v_minus)
    {
 return v_origin - v_minus;
    }
 else
    {
 return v_origin + (wrap - v_minus);
    }
}
/**************************************************************************
```

```
 * End of function apl_mem_usage_update
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : demo_thread_entry
 * Description : Printer Demo Thread
 * Arguments : uint32_t thread_input
 * Return value : none
 ***********************************************************************************/
void demo_thread_entry (uint32_t thread_input)
{
    uint32_t tick0, tick1, diff;
    uint32_t pmem       = 0;
    uint8_t port_status = device_printer_port_status.value;
 /* Not currently using thread_input. */
 FSP_PARAMETER_NOT_USED(thread_input);
 /* Standalone stack: run tasks in application thread loop. */
 while (1)
    {
 /* Update memory usage. */
       apl_mem_usage_update();
 /* Let other threads to run. */
       tx_thread_sleep(1);
 /* Do status change check. */
 if (device_printer &&
           (port_status != device_printer_port_status.value))
     {
        ux_device_class_printer_ioctl(device_printer,
                                      UX_DEVICE_CLASS_PRINTER_IOCTL_PORT_STATUS_SET,
                                      (void *) device_printer_port_status.value);
           port_status = device_printer_port_status.value;
     }
 /* Check time passed and update speed every 1s. */
       tick1 = tx_time_get();
       diff = apl_minus(tick1, tick0, 0xFFFFFFFF);
```

```
 if (diff < TX_TIMER_TICKS_PER_SECOND)
      {
 continue;
      }
       tick0 = tick1;
 /* Print results. */
 if ((pmem != test_data.mem_usage_max))
      {
            pmem = test_data.mem_usage_max;
      }
    }
}
/**************************************************************************
 * End of function demo_thread_entry
 **************************************************************************/
/**************************************************************************
 * Function Name : printer_read_thread_entry
 * Description : USB read operation and echo back the first part of user input on
serial terminal.
 * Arguments : uint32_t thread_input
 * Return value : none
 **************************************************************************/
void printer_read_thread_entry (uint32_t thread_input)
{
    UINT    status;
    uint32_t actual_length;
    UINT    i;
 /* Not currently using thread_input. */
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
 if (device_printer == UX_NULL)
      {
 /* Wait a while before next check. */
```

```
        tx_thread_sleep(10);
 continue;
     }
      status = ux_device_class_printer_read(device_printer, device_printer_buffer,
DATA_LEN, &actual_length);
 if (status != UX_SUCCESS)
     {
 continue;
     }
 if (actual_length == 0)
     {
 continue;
     }
   }
}
/**********************************************************************************
 * End of function printer_read_thread_entry
 **********************************************************************************/
/**********************************************************************************
 * Function Name : printer_write_thread_entry
 * Description : Periodically checks the printer status.
 * Arguments : uint32_t thread_input
 * Return value : none
 **********************************************************************************/
void printer_write_thread_entry (uint32_t thread_input)
{
    uint8_t port_status = device_printer_port_status.value;
 /* Not currently using thread_input. */
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
 /* Wait 2s. */
      tx_thread_sleep(TX_TIMER_TICKS_PER_SECOND * 2);
 if (device_printer &&
```

```
            (port_status != device_printer_port_status.value))
     {
 /* Send status and other information here. */
            port_status = device_printer_port_status.value;
     }
   }
}
```

### USBX HPRN Example

HPRN example is as follows.

```
#include "r_usb_basic.h"
#include "r_usb_basic_cfg.h"
#include "ux_api.h"
#include "ux_system.h"
#include "ux_host_class_printer.h"
/*******************************************************************************
 * Macro definitions
 ******************************************************************************/
#define DATA_LEN (1030U)
#define MAX_REQUEST_SIZE (2048U)
#define MEMPOOL_SIZE (18432)
#define HPRN_FLAG ((uint32_t) 0x0001)
#define VALUE_4 (4)
#define MOD_VAL (50)
#define READ_LEN (64)
#define WAIT_TIME (50)
#define SUCCESS (0U)
#define UX_FSP_DEVICE_INSERTION (0x01U)
#define UX_FSP_DEVICE_REMOVAL (0x02U)
#define RESET_VALUE (0x00)
/* Private function */
static UINT ux_host_usr_event_notification(ULONG event, UX_HOST_CLASS * host_class,
VOID * instance);
```

```
/* A pointer to store Printer instance. */
static UX_HOST_CLASS_PRINTER * p_printer = UX_NULL;
static ULONG g_write_actual_length = RESET_VALUE;
static ULONG g_read_actual_length = RESET_VALUE;
static uint8_t g_read_buf[DATA_LEN] = {RESET_VALUE};
static uint8_t g_read_buf1[DATA_LEN] = {RESET_VALUE};
static uint8_t g_write_buf[DATA_LEN] = {RESET_VALUE};
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / VALUE_4];
/* HPRN Thread entry function */
void hprn_thread_entry (void)
{
    uint32_t status       = RESET_VALUE;
    ULONG    actual_flags = RESET_VALUE;
    uint16_t count        = RESET_VALUE;
    ULONG    port_status = RESET_VALUE;
 /* ux_system_initialization */
    ux_system_initialize(g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, RESET_VALUE);
 /* ux host stack initialization */
    ux_host_stack_initialize(ux_host_usr_event_notification);
 /* Open usb driver */
 R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 /*Fill the write buffer*/
 for (count = RESET_VALUE; count < DATA_LEN; count++)
    {
       g_write_buf[count] = (uint8_t) count;
    }
 while (true)
    {
 /* retrieves event flags from the specified event flags group.*/
       tx_event_flags_get(&g_printer_activate_event_flags0, HPRN_FLAG, TX_OR,
&actual_flags, TX_WAIT_FOREVER);
 if (UX_NULL != p_printer)
     {
 /* GET_PORT_STATUS */
```

```
            ux_host_class_printer_status_get(p_printer, &port_status);
 /*Send the data to device*/
            ux_host_class_printer_write(p_printer, g_write_buf, DATA_LEN,
&g_write_actual_length);
 /* Clear the buffer */
            memset(g_read_buf, RESET_VALUE, sizeof(g_read_buf));
 /* USB receives the data echoed back */
            ux_host_class_printer_read(p_printer, g_read_buf, MAX_REQUEST_SIZE,
&g_read_actual_length);
 /*compare loop-back data*/
 if (SUCCESS != (memcmp(g_read_buf, g_write_buf, sizeof(g_read_buf))))
      {
 return;
      }
            tx_thread_sleep(WAIT_TIME);
      }
    }
}
static UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class,
VOID * instance)
{
 if (_ux_utility_memory_compare(_ux_system_host_class_printer_name, host_class,

_ux_utility_string_length_get(_ux_system_host_class_printer_name)) ==
       UX_SUCCESS)
    {
 /* Check if there is a device insertion. */
 if (UX_FSP_DEVICE_INSERTION == event)
      {
            p_printer = (UX_HOST_CLASS_PRINTER *) instance;
 if (UX_NULL != p_printer)
      {
 /* This sets or clears event flags in an event flags group */
               tx_event_flags_set(&g_printer_activate_event_flags0, HPRN_FLAG,
```

```
TX_OR);
     }
     }
 /* Check if there is a device removal */
 else if (UX_FSP_DEVICE_REMOVAL == event)
     {
 /* This sets or clears event flags in an event flags group */
           tx_event_flags_set(&g_printer_activate_event_flags0, ~HPRN_FLAG, TX_AND);
           p_printer = UX_NULL;
     }
 else
     {
 /*do nothing */
     }
   }
 return UX_SUCCESS;
}
```

### USBX HUVC Example

HUVC example is as follows.

```
/**********************************************************************************************
**********************************
 * Macro definitions
 **********************************************************************************************
**********************************/
 #define RESET_VALUE (0x00)
 #define MEMPOOL_SIZE (18432)
 #define VALUE_4 (4)
 #define EVENTFLAG_USB_DEVICE_INSERTED (0x01)
/* Define the number of buffers used in this demo. */
 #define MAX_NUM_BUFFERS 2
 #define USBFS_ISO_PIPE_MAX_PAKCET_SIZE (256)
/**********************************************************************************************
```

```
**********************************
 * Private function prototypes
 ***************************************************************************************
**********************************/
VOID uvc_transfer_request_done_callback(UX_TRANSFER * transfer_request);
VOID uvc_parameter_interval_list(UX_HOST_CLASS_VIDEO * video);
UINT uvc_parameter_frame_list(UX_HOST_CLASS_VIDEO * video);
VOID uvc_parameter_list(UX_HOST_CLASS_VIDEO * video);
VOID uvc_process_function(UX_HOST_CLASS_VIDEO * video);
UINT ux_host_usr_event_notification(ULONG event, UX_HOST_CLASS * host_class, VOID *
instance);
/***************************************************************************************
**********************************
 * Private global variables
 ***************************************************************************************
**********************************/
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / VALUE_4];
/* video class instance */
UX_HOST_CLASS_VIDEO * volatile video_host_class;
TX_EVENT_FLAGS_GROUP g_device_insert_eventflag;
TX_SEMAPHORE         g_data_received_semaphore;
/* video buffer */
UCHAR g_video_buffer[10 * 1024];
/* Name string of VS types */
struct
{
 int    type;
 char * name;
} vs_type_name[] =
{
    {UX_HOST_CLASS_VIDEO_VS_UNDEFINED, "UX_HOST_CLASS_VIDEO_VS_UNDEFINED"     },
    {UX_HOST_CLASS_VIDEO_VS_INPUT_HEADER, "UX_HOST_CLASS_VIDEO_VS_INPUT_HEADER"
},
    {UX_HOST_CLASS_VIDEO_VS_OUTPUT_HEADER, "UX_HOST_CLASS_VIDEO_VS_OUTPUT_HEADER"
```

```
},
    {UX_HOST_CLASS_VIDEO_VS_STILL_IMAGE_FRAME,
"UX_HOST_CLASS_VIDEO_VS_STILL_IMAGE_FRAME"     },
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_UNCOMPRESSED,
"UX_HOST_CLASS_VIDEO_VS_FORMAT_UNCOMPRESSED"     },
    {UX_HOST_CLASS_VIDEO_VS_FRAME_UNCOMPRESSED,
"UX_HOST_CLASS_VIDEO_VS_FRAME_UNCOMPRESSED"     },
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_MJPEG, "UX_HOST_CLASS_VIDEO_VS_FORMAT_MJPEG"
},
    {UX_HOST_CLASS_VIDEO_VS_FRAME_MJPEG, "UX_HOST_CLASS_VIDEO_VS_FRAME_MJPEG"     },
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_MPEG2TS, "UX_HOST_CLASS_VIDEO_VS_FORMAT_MPEG2TS"
},
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_DV, "UX_HOST_CLASS_VIDEO_VS_FORMAT_DV"     },
    {UX_HOST_CLASS_VIDEO_VS_COLORFORMAT, "UX_HOST_CLASS_VIDEO_VS_COLORFORMAT"     },
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_FRAME_BASED,
"UX_HOST_CLASS_VIDEO_VS_FORMAT_FRAME_BASED"     },
    {UX_HOST_CLASS_VIDEO_VS_FRAME_FRAME_BASED,
"UX_HOST_CLASS_VIDEO_VS_FRAME_FRAME_BASED"     },
    {UX_HOST_CLASS_VIDEO_VS_FORMAT_STREAM_BASED,
"UX_HOST_CLASS_VIDEO_VS_FORMAT_STREAM_BASED"     }
};
/* HUVC Thread entry function */
void usbx_huvc_sample (void)
{
    uint32_t status       = RESET_VALUE;
    ULONG    actual_flags = RESET_VALUE;
 fsp_err_t err          = FSP_SUCCESS;
 /* ux_system_initialization */
    status = ux_system_initialize(g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL,
RESET_VALUE);
 if (UX_SUCCESS != status)
    {
 while (1)
     {
```

```
            ;
      }
   }
/* ux host stack initialization */
   status = ux_host_stack_initialize(ux_host_usr_event_notification);
if (UX_SUCCESS != status)
   {
while (1)
     {
            ;
     }
   }
/* Open usb driver */
   err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
if (FSP_SUCCESS != err)
   {
while (1)
     {
            ;
     }
   }
   tx_event_flags_create(&g_device_insert_eventflag, (CHAR *) "Device Insert Event
Flags");
   tx_semaphore_create(&g_data_received_semaphore, "payload semaphore", 0);
while (1)
   {
/* Suspend here until a USBX Host Class Instance gets ready. */
      tx_event_flags_get(&g_device_insert_eventflag,
                         EVENTFLAG_USB_DEVICE_INSERTED,
                         TX_OR,
                         (ULONG *) &actual_flags,
                         TX_WAIT_FOREVER);
/* This delay is required for now to get valid ISO IN UX_ENDPOINT instance. */
      tx_thread_sleep(100);
```

```
 if (UX_NULL != video_host_class)
      {
            uvc_process_function(video_host_class);
      }
    }
}
/* USBX Host event notification callback function */
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
 if (UX_SUCCESS ==
       _ux_utility_memory_compare(_ux_system_host_class_video_name, host_class,

_ux_utility_string_length_get(_ux_system_host_class_video_name)))
    {
 if (UX_DEVICE_INSERTION == event) /* Check if there is a device insertion. */
      {
            video_host_class = instance;
 /* Set the event flag to let application know the device insertion. */
            tx_event_flags_set(&g_device_insert_eventflag,
EVENTFLAG_USB_DEVICE_INSERTED, TX_OR);
      }
 else if (UX_DEVICE_REMOVAL == event)
      {
 /* Clear the event flag in case the camera was removed before the application could
clear it. */
            tx_event_flags_set(&g_device_insert_eventflag, (ULONG)
~EVENTFLAG_USB_DEVICE_INSERTED, TX_AND);
            video_host_class = NULL;
      }
    }
 return UX_SUCCESS;
}
/* Video data received callback function. */
```

```
VOID uvc_transfer_request_done_callback (UX_TRANSFER * transfer_request)
{
 /* This is the callback function invoked by UVC class after a packet of
  * data is received. */
 /* The actual number of bytes being received into the data buffer is
  * recorded in tranfer_request -> ux_transfer_request_actual_length. */
 /* Since this callback function executes in the USB host controller
  * thread, a semaphore is released so the application can pick up the
  * video data in application thread. */
 FSP_PARAMETER_NOT_USED(transfer_request);
    tx_semaphore_put(&g_data_received_semaphore);
}
/* Show the interval types */
VOID uvc_parameter_interval_list (UX_HOST_CLASS_VIDEO * video)
{
    UX_HOST_CLASS_VIDEO_FRAME_DESCRIPTOR frame_descriptor;
    ULONG min_frame_interval;
    ULONG max_frame_interval;
    ULONG frame_interval_step;
 int   i;
 /* Make the descriptor machine independent. */
    _ux_utility_descriptor_parse(video->ux_host_class_video_current_frame_address,
                                 _ux_system_class_video_frame_descriptor_structure,
                                 UX_HOST_CLASS_VIDEO_FRAME_DESCRIPTOR_ENTRIES,
                                 (UCHAR *) &frame_descriptor);
 /* Check the frame interval type. */
 if (0 == frame_descriptor.bFrameIntervalType)
    {
 /* Frame interval type is continuous. */
       min_frame_interval =
_ux_utility_long_get(video->ux_host_class_video_current_frame_address + 26);
       max_frame_interval =
_ux_utility_long_get(video->ux_host_class_video_current_frame_address + 30);
       frame_interval_step =
```

```
_ux_utility_long_get(video->ux_host_class_video_current_frame_address + 34);
 FSP_PARAMETER_NOT_USED(min_frame_interval);
 FSP_PARAMETER_NOT_USED(max_frame_interval);
 FSP_PARAMETER_NOT_USED(frame_interval_step);
    }
}
/* Show the frame resolutions */
UINT uvc_parameter_frame_list (UX_HOST_CLASS_VIDEO * video)
{
    ULONG frame_index;
    UX_HOST_CLASS_VIDEO_PARAMETER_FRAME_DATA frame_parameter;
    UINT status = UX_SUCCESS;
 /* frame resolutions */
 for (frame_index = 1; frame_index <= video->ux_host_class_video_number_frames;
frame_index++)
    {
 /* Get frame data for current frame index. */
       frame_parameter.ux_host_class_video_parameter_frame_requested = frame_index;
       status = _ux_host_class_video_frame_data_get(video, &frame_parameter);
 if (UX_SUCCESS != status)
     {
 return status;
     }
 /* Save the current frame index. */
       video->ux_host_class_video_current_frame = frame_index;
       uvc_parameter_interval_list(video);
    }
 return status;
}
/* Show the device parameters */
VOID uvc_parameter_list (UX_HOST_CLASS_VIDEO * video)
{
    ULONG format_index;
    UX_HOST_CLASS_VIDEO_PARAMETER_FORMAT_DATA format_parameter;
```

```
    UINT status = UX_SUCCESS;
 int i;
 /* format types */
 for (format_index = 1; format_index <= video->ux_host_class_video_number_formats;
format_index++)
    {
 /* Get format data for current format index. */
       format_parameter.ux_host_class_video_parameter_format_requested =
format_index;
       status = _ux_host_class_video_format_data_get(video, &format_parameter);
 if (UX_SUCCESS == status)
     {
 /* Save number of frames in the video instance. */
           video->ux_host_class_video_number_frames =

format_parameter.ux_host_class_video_parameter_number_frame_descriptors;
           uvc_parameter_frame_list(video);
     }
    }
}
VOID uvc_process_function (UX_HOST_CLASS_VIDEO * video)
{
 /* This demo uses two buffers. One buffer is used by video device while the
  * application consumes data in the other buffer. */
    UCHAR * buffer_ptr[MAX_NUM_BUFFERS];
 /* Index variable keeping track of the current buffer being used by the video
device. */
    ULONG buffer_index;
 /* Maximum buffer requirement reported by the video device. */
    ULONG max_buffer_size;
    UINT status;
    ULONG actual_flags;
    UINT frame_count;
    UX_HOST_CLASS_VIDEO_PARAMETER_CHANNEL channel;
```

```
 /* List parameters */
    uvc_parameter_list(video);
 /* Set video parameters. This setting value is a dummy.
  * Depending on the application, set the necessary parameters. */
    status = ux_host_class_video_frame_parameters_set(video,
UX_HOST_CLASS_VIDEO_VS_FORMAT_MJPEG, 176, 144, 333333);
 /* Set the user callback function of video class. */
    ux_host_class_video_transfer_callback_set(video,
uvc_transfer_request_done_callback);
 /* Find out the maximum memory buffer size for the video configuration
  * set above. */
    max_buffer_size = ux_host_class_video_max_payload_get(video);
 /* USBFS's Max Packet Size is 256 */
 if (0 == g_basic0_cfg.module_number) /* 0 : USBFS */
    {
 if (max_buffer_size > USBFS_ISO_PIPE_MAX_PAKCET_SIZE)
      {
            max_buffer_size = USBFS_ISO_PIPE_MAX_PAKCET_SIZE;
      }
    }
 /* Clear semaphore to zero */
 while (1)
    {
 if (TX_NO_INSTANCE == tx_semaphore_get(&g_data_received_semaphore, 0))
      {
 break;
      }
    }
 if (0 == g_basic0_cfg.module_number) /* 0 : USBFS */
    {
       video->ux_host_class_video_transfer_request_start_index = 0;
       video->ux_host_class_video_transfer_request_end_index   = 0;
       channel.ux_host_class_video_parameter_format_requested         =
video->ux_host_class_video_current_format;
```

```
      channel.ux_host_class_video_parameter_frame_requested          =
video->ux_host_class_video_current_frame;
      channel.ux_host_class_video_parameter_frame_interval_requested =
           video->ux_host_class_video_current_frame_interval;
      channel.ux_host_class_video_parameter_channel_bandwidth_selection =
max_buffer_size;
      status = ux_host_class_video_ioctl(video,
UX_HOST_CLASS_VIDEO_IOCTL_CHANNEL_START, &channel);
   }
 else
   {
 /* Start video transfer. */
      status = ux_host_class_video_start(video);
 if (UX_SUCCESS != status)
     {
 /* Setting these to zero is a hack since we're mixing old and new APIs (new API does
this and is required for reads). */
        video->ux_host_class_video_transfer_request_start_index = 0;
        video->ux_host_class_video_transfer_request_end_index   = 0;
           channel.ux_host_class_video_parameter_format_requested =
               video->ux_host_class_video_current_format;
           channel.ux_host_class_video_parameter_frame_requested =
               video->ux_host_class_video_current_frame;
        channel.ux_host_class_video_parameter_frame_interval_requested =
             video->ux_host_class_video_current_frame_interval;
           channel.ux_host_class_video_parameter_channel_bandwidth_selection = 1024;
        status = ux_host_class_video_ioctl(video,
UX_HOST_CLASS_VIDEO_IOCTL_CHANNEL_START, &channel);
     }
   }
 /* Allocate space for video buffer. */
 for (buffer_index = 0; buffer_index < MAX_NUM_BUFFERS; buffer_index++)
   {
      buffer_ptr[buffer_index] = &g_video_buffer[max_buffer_size * buffer_index];
```

```
   }
   buffer_index = 0;
   frame_count = 0;
while (1)
   {
/* Add the buffer back for video transfer. */
      ux_host_class_video_transfer_buffer_add(video, buffer_ptr[buffer_index]);
/* Increment the buffer_index, and wrap to zero if it exceeds the
 * maximum number of buffers. */
      buffer_index = (buffer_index + 1);
if (buffer_index >= MAX_NUM_BUFFERS)
     {
     buffer_index = 0;
     }
/* Suspend here until a transfer callback is called. */
      status = tx_semaphore_get(&g_data_received_semaphore, 100);
if (TX_SUCCESS != status)
     {
/* Check camera status */
           status = tx_event_flags_get(&g_device_insert_eventflag,
                            EVENTFLAG_USB_DEVICE_INSERTED,
                            TX_OR,
                            (ULONG *) &actual_flags,
                                      0);
if (TX_SUCCESS == status)
     {
/* Stop video transfer. */
               ux_host_class_video_stop(video);
     }
break;
     }
/* Received data. The callback function needs to obtain the actual
 * number of bytes received, so the application routine can read the
 * correct amount of data from the buffer. */
```

```
 /* Application can now consume video data while the video device stores
  * the data into the other buffer. */
       frame_count++;
    }
}
```

### USBX HAUD Example

HAUD example is as follows.

```
/**********************************************************************************
*********************************
 * Macro definitions
 **********************************************************************************
*********************************/
 #define RESET_VALUE (0x00)
 #define MEMPOOL_SIZE (18432)
 #define VALUE_4 (4)
 #define EVENTFLAG_USB_DEVICE_INSERTED (0x01)
/* Define the number of buffers used in this demo. */
 #define MAX_NUM_BUFFERS 2
 #define USBFS_ISO_PIPE_MAX_PAKCET_SIZE (256)
/**********************************************************************************
*********************************
 * Private function prototypes
 **********************************************************************************
*********************************/
VOID usbx_haudio_transfer_request_done_callback(UX_TRANSFER * transfer_request);
VOID usbx_haudio_parameter_interval_list(UX_HOST_CLASS_AUDIO * audio);
UINT usbx_haudio_parameter_frame_list(UX_HOST_CLASS_AUDIO * audio);
VOID usbx_haudio_parameter_list(UX_HOST_CLASS_AUDIO * audio);
VOID usbx_haudio_process_function(UX_HOST_CLASS_AUDIO * audio);
UINT ux_host_usr_event_notification(ULONG event, UX_HOST_CLASS * host_class, VOID *
instance);
/**********************************************************************************
```

```
**********************************
 * Private global variables
 ***************************************************************************************
**********************************/
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / VALUE_4];
audio class instance * /
UX_HOST_CLASS_AUDIO * volatile audio_host_class;
TX_EVENT_FLAGS_GROUP g_device_insert_eventflag;
TX_SEMAPHORE         g_data_received_semaphore;
/* audio buffer */
UCHAR g_audio_buffer[10 * 1024];
/* HAUD Thread entry function */
void usbx_haud_sample (void)
{
    uint32_t status       = RESET_VALUE;
    ULONG    actual_flags = RESET_VALUE;
 fsp_err_t err          = FSP_SUCCESS;
 /* ux_system_initialization */
    status = ux_system_initialize(g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL,
RESET_VALUE);
 if (UX_SUCCESS != status)
    {
 while (1)
     {
           ;
     }
    }
 /* ux host stack initialization */
    status = ux_host_stack_initialize(ux_host_usr_event_notification);
 if (UX_SUCCESS != status)
    {
 while (1)
     {
           ;
```

```
      }
    }
 /* Open usb driver */
    err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
 while (1)
      {
            ;
      }
    }
    tx_event_flags_create(&g_device_insert_eventflag, (CHAR *) "Device Insert Event
Flags");
    tx_semaphore_create(&g_data_received_semaphore, "payload semaphore", 0);
 while (1)
    {
 /* Suspend here until a USBX Host Class Instance gets ready. */
       tx_event_flags_get(&g_device_insert_eventflag,
                          EVENTFLAG_USB_DEVICE_INSERTED,
                          TX_OR,
                          (ULONG *) &actual_flags,
                          TX_WAIT_FOREVER);
 /* This delay is required for now to get valid ISO IN UX_ENDPOINT instance. */
       tx_thread_sleep(100);
 if (UX_NULL != audio_host_class)
      {
            usbx_haudio_process_function(audio_host_class);
      }
    }
}
/* USBX Host event notification callback function */
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
```

```
 if (UX_SUCCESS ==
       _ux_utility_memory_compare(_ux_system_host_class_audio_name, host_class,

_ux_utility_string_length_get(_ux_system_host_class_audio_name)))
    {
 if (UX_DEVICE_INSERTION == event) /* Check if there is a device insertion. */
      {
            audio_host_class = instance;
 /* Set the event flag to let application know the device insertion. */
            tx_event_flags_set(&g_device_insert_eventflag,
EVENTFLAG_USB_DEVICE_INSERTED, TX_OR);
      }
 else if (UX_DEVICE_REMOVAL == event)
      {
 /* Clear the event flag in case the camera was removed before the application could
clear it. */
            tx_event_flags_set(&g_device_insert_eventflag, (ULONG)
~EVENTFLAG_USB_DEVICE_INSERTED, TX_AND);
            audio_host_class = NULL;
      }
    }
 return UX_SUCCESS;
}
/* Video data received callback function. */
VOID usbx_haudio_transfer_request_done_callback (UX_TRANSFER * transfer_request)
{
 /* This is the callback function invoked by Audio class after a packet of
  * data is received. */
 /* The actual number of bytes being received into the data buffer is
  * recorded in tranfer_request -> ux_transfer_request_actual_length. */
 /* Since this callback function executes in the USB host controller
  * thread, a semaphore is released so the application can pick up the
  * audio data in application thread. */
 FSP_PARAMETER_NOT_USED(transfer_request);
```

```
    tx_semaphore_put(&g_data_received_semaphore);
}
/* Show the interval types */
VOID usbx_haudio_parameter_interval_list (UX_HOST_CLASS_AUDIO * audio; )
{
    UX_HOST_CLASS_AUDIO_FRAME_DESCRIPTOR frame_descriptor;
    ULONG min_frame_interval;
    ULONG max_frame_interval;
    ULONG frame_interval_step;
 int   i;
 /* Make the descriptor machine independent. */
    _ux_utility_descriptor_parse(audio->ux_host_class_audio_current_frame_address,
                                 _ux_system_class_audio_frame_descriptor_structure,
                                 UX_HOST_CLASS_AUDIO_FRAME_DESCRIPTOR_ENTRIES,
                                 (UCHAR *) &frame_descriptor);
 /* Check the frame interval type. */
 if (0 == frame_descriptor.bFrameIntervalType)
    {
 /* Frame interval type is continuous. */
       min_frame_interval =
_ux_utility_long_get(audio->ux_host_class_audio_current_frame_address + 26);
       max_frame_interval =
_ux_utility_long_get(audio->ux_host_class_audio_current_frame_address + 30);
       frame_interval_step =
_ux_utility_long_get(audio->ux_host_class_audio_current_frame_address + 34);
 FSP_PARAMETER_NOT_USED(min_frame_interval);
 FSP_PARAMETER_NOT_USED(max_frame_interval);
 FSP_PARAMETER_NOT_USED(frame_interval_step);
    }
}
/* Show the frame resolutions */
UINT usbx_haudio_parameter_frame_list (UX_HOST_CLASS_AUDIO * audio)
{
    ULONG frame_index;
```

```
    UX_HOST_CLASS_AUDIO_PARAMETER_FRAME_DATA frame_parameter;
    UINT status = UX_SUCCESS;
 /* frame resolutions */
 for (frame_index = 1; frame_index <= audio->ux_host_class_audio_number_frames;
frame_index++)
    {
 /* Get frame data for current frame index. */
       frame_parameter.ux_host_class_audio_parameter_frame_requested = frame_index;
       status = _ux_host_class_audio_frame_data_get(audio, &frame_parameter);
 if (UX_SUCCESS != status)
     {
 return status;
     }
 /* Save the current frame index. */
       audio->ux_host_class_audio_current_frame = frame_index;
       usbx_haudio_parameter_interval_list(audio);
    }
 return status;
}
/* Show the device parameters */
VOID usbx_haudio_parameter_list (UX_HOST_CLASS_AUDIO * audio)
{
    ULONG format_index;
    UX_HOST_CLASS_AUDIO_PARAMETER_FORMAT_DATA format_parameter;
    UINT status = UX_SUCCESS;
 int i;
 /* format types */
 for (format_index = 1; format_index <= audio->ux_host_class_audio_number_formats;
format_index++)
    {
 /* Get format data for current format index. */
       format_parameter.ux_host_class_audio_parameter_format_requested =
format_index;
       status = _ux_host_class_audio_format_data_get(audio, &format_parameter);
```

```
 if (UX_SUCCESS == status)
      {
 /* Save number of frames in the audio instance. */
            audio->ux_host_class_audio_number_frames =

format_parameter.ux_host_class_audio_parameter_number_frame_descriptors;
            usbx_haudio_parameter_frame_list(audio);
      }
    }
}
VOID usbx_haudio_process_function (UX_HOST_CLASS_AUDIO * audio)
{
 /* This demo uses two buffers. One buffer is used by audio device while the
  * application consumes data in the other buffer. */
    UCHAR * buffer_ptr[MAX_NUM_BUFFERS];
 /* Index variable keeping track of the current buffer being used by the audio
device. */
    ULONG buffer_index;
 /* Maximum buffer requirement reported by the audio device. */
    ULONG max_buffer_size;
    UINT status;
    ULONG actual_flags;
    UINT frame_count;
    UX_HOST_CLASS_AUDIO_PARAMETER_CHANNEL channel;
 /* List parameters */
    usbx_haudio_parameter_list(audio);
 /* Set audio parameters. This setting value is a dummy.
  * Depending on the application, set the necessary parameters. */
    status = ux_host_class_audio_frame_parameters_set(audio,
UX_HOST_CLASS_AUDIO_VS_FORMAT_MJPEG, 176, 144, 333333);
 /* Set the user callback function of audio class. */
    ux_host_class_audio_transfer_callback_set(audio,
usbx_haudio_transfer_request_done_callback);
 /* USBFS's Max Packet Size is 256 */
```

```
 if (0 == g_basic0_cfg.module_number) /* 0 : USBFS */
    {
 if (max_buffer_size > USBFS_ISO_PIPE_MAX_PAKCET_SIZE)
      {
            max_buffer_size = USBFS_ISO_PIPE_MAX_PAKCET_SIZE;
      }
    }
 /* Clear semaphore to zero */
 while (1)
    {
 if (TX_NO_INSTANCE == tx_semaphore_get(&g_data_received_semaphore, 0))
      {
 break;
      }
    }
 if (0 == g_basic0_cfg.module_number) /* 0 : USBFS */
    {
       audio->ux_host_class_audio_transfer_request_start_index = 0;
       audio->ux_host_class_audio_transfer_request_end_index   = 0;
       channel.ux_host_class_audio_parameter_format_requested         =
audio->ux_host_class_audio_current_format;
       channel.ux_host_class_audio_parameter_frame_requested          =
audio->ux_host_class_audio_current_frame;
       channel.ux_host_class_audio_parameter_frame_interval_requested =
            audio->ux_host_class_audio_current_frame_interval;
       channel.ux_host_class_audio_parameter_channel_bandwidth_selection =
max_buffer_size;
       status = ux_host_class_audio_ioctl(audio,
UX_HOST_CLASS_AUDIO_IOCTL_CHANNEL_START, &channel);
    }
 else
    {
 /* Start audio transfer. */
       status = ux_host_class_audio_start(audio);
```

```
 if (UX_SUCCESS != status)
      {
 /* Setting these to zero is a hack since we're mixing old and new APIs (new API does
this and is required for reads). */
         audio->ux_host_class_audio_transfer_request_start_index = 0;
         audio->ux_host_class_audio_transfer_request_end_index   = 0;
            channel.ux_host_class_audio_parameter_format_requested =
                audio->ux_host_class_audio_current_format;
            channel.ux_host_class_audio_parameter_frame_requested =
                audio->ux_host_class_audio_current_frame;
         channel.ux_host_class_audio_parameter_frame_interval_requested =
              audio->ux_host_class_audio_current_frame_interval;
            channel.ux_host_class_audio_parameter_channel_bandwidth_selection = 1024;
         status = ux_host_class_audio_ioctl(audio,
UX_HOST_CLASS_AUDIO_IOCTL_CHANNEL_START, &channel);
      }
    }
 /* Allocate space for audio buffer. */
 for (buffer_index = 0; buffer_index < MAX_NUM_BUFFERS; buffer_index++)
    {
       buffer_ptr[buffer_index] = &g_audio_buffer[max_buffer_size * buffer_index];
    }
    buffer_index = 0;
    frame_count = 0;
 while (1)
    {
 /* Increment the buffer_index, and wrap to zero if it exceeds the
  * maximum number of buffers. */
       buffer_index = (buffer_index + 1);
 if (buffer_index >= MAX_NUM_BUFFERS)
      {
      buffer_index = 0;
      }
 /* Suspend here until a transfer callback is called. */
```

```
      status = tx_semaphore_get(&g_data_received_semaphore, 100);
 if (TX_SUCCESS != status)
     {
 /* Check camera status */
           status = tx_event_flags_get(&g_device_insert_eventflag,
                              EVENTFLAG_USB_DEVICE_INSERTED,
                              TX_OR,
                              (ULONG *) &actual_flags,
                                        0);
 if (TX_SUCCESS == status)
     {
 /* Stop audio transfer. */
               ux_host_class_audio_stop(audio);
     }
 break;
     }
 /* Received data. The callback function needs to obtain the actual
  * number of bytes received, so the application routine can read the
  * correct amount of data from the buffer. */
 /* Application can now consume audio data while the audio device stores
  * the data into the other buffer. */
      frame_count++;
   }
}
```

### USBX DFU Example

DFU example is as follows.

```
/***********************************************************************
 * Macro definitions
 ***********************************************************************/
#define CDCACM_FLAG ((ULONG) 0x0001)
#define CDCACM_ACTIVATE_FLAG ((ULONG) 0x0004)
#define CDCACM_DEACTIVATE_FLAG ((ULONG) 0x0008)
```

```
#define DFU_FLAG ((ULONG) 0x0001)
#define DFU_ACTIVATE_FLAG ((ULONG) 0x0004)
#define DFU_DEACTIVATE_FLAG ((ULONG) 0x0008)
#define DFU_DETACH_REQUEST_FLAG ((ULONG) 0x0200)
#define RESET_VALUE (0U)
#define CONFIG_NUMB1 (1U)
#define CONFIG_NUMB0 (0U)
#define INTERFACE_NUMB0 (0U)
#define INTERFACE_NUMB1 (1U)
#define INTERFACE_NUMB2 (2U)
#define DFU_DEVICE_FRAME_LENGTH_FULL_SPEED (45U)
#define DFU_STRING_FRAMEWORK_LENGTH (94U)
#define DFU_FIRM_UPDATE_MAX_TRY (1U)
/**************************************************************************
 * Exported global variables and functions (to be accessed by other files)
 *************************************************************************/
extern uint8_t g_device_framework_full_speed[];
extern uint8_t g_device_framework_hi_speed[];
extern uint8_t g_language_id_framework[];
extern uint8_t g_string_framework[];
extern uint8_t g_dfu_device_framework_full_speed[];
extern uint8_t g_dfu_string_framework[];
extern uint32_t usb_peri_usbx_initialize(uint32_t dcd_io);
/**************************************************************************
 * Global functions and variables
 *************************************************************************/
UINT usbx_status_callback(ULONG status);
/**************************************************************************
 * Private global variables and functions
 *************************************************************************/
static void ux_cdc_device0_instance_activate(void * cdc_instance);
static void ux_cdc_device0_instance_deactivate(void * cdc_instance);
static void usbx_pcdc_operations(void);
/* Mempool size of 18k is required for USBX device class pre built libraries
```

```
 * and it is valid only if it with default USBX configurations. */
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / BYTE_SIZE];
static UX_SLAVE_CLASS_CDC_ACM_PARAMETER g_ux_device_class_cdc_acm0_parameter;
static UX_SLAVE_CLASS_CDC_ACM * g_cdc;
static ULONG g_actual_length;
static uint8_t g_buf[DATA_LEN];
static UX_SLAVE_CLASS_DFU_PARAMETER g_ux_device_class_dfu_parameter;
static UX_SLAVE_CLASS_DFU * g_dfu;
static UINT g_dfu_firmware_update_done_count = 0;
static void dfu_register_function(UINT if_num);
static void mode_change_dfu_to_cdc(void);
static void mode_change_cdc_to_dfu(void);
static UINT dfu_dammy_write(VOID * dfu, ULONG block_number, UCHAR * data_pointer,
ULONG length, ULONG * media_status);
static UINT dfu_dammy_get_status(VOID * dfu, ULONG * media_status);
static UINT dfu_dammy_notify(VOID * dfu, ULONG notification);
/* PCDC ACM & DFU Thread entry function */
void pcdc_dfu_thread_entry (void)
{
 /* To check ux api return status */
    UINT status = UX_SUCCESS;
 /* To check fsp api return status */
 fsp_err_t err          = FSP_SUCCESS;
    ULONG    actual_flags = 0x0000;
    UX_SLAVE_CLASS_DFU * dfu;
    UCHAR                state;
 /* ux_system_initialization */
    status = ux_system_initialize(g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL,
RESET_VALUE);
 /* ux_device stack initialization */
    status = ux_device_stack_initialize(g_device_framework_hi_speed,
                                DEVICE_FRAME_LENGTH_HIGH_SPEED,
                                g_device_framework_full_speed,
                                DEVICE_FRAME_LENGTH_FULL_SPEED,
```

```
                                g_string_framework,
                                STRING_FRAMEWORK_LENGTH,
                                g_language_id_framework,
                                LANGUAGE_ID_FRAME_WORK_LENGTH,
                                      &usbx_status_callback);
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate   =
ux_cdc_device0_instance_activate;
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
       ux_cdc_device0_instance_deactivate;
 /* ux_device stack class registration */
    status = ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                   _ux_device_class_cdc_acm_entry,
                                          CONFIG_NUMB1,
                                   INTERFACE_NUMB0,
                                   (void *) &g_ux_device_class_cdc_acm0_parameter);
 /* ux_device stack class registration (DFU)*/
    dfu_register_function(INTERFACE_NUMB2); /* Input : IF number */
 /* Open usb driver */
    err = R_USB_Open(&g_basic1_ctrl, &g_basic1_cfg);
 /* wait for enumeration event */
    status = tx_event_flags_get(&g_cdcacm_event_flags0, CDCACM_ACTIVATE_FLAG, TX_OR,
&actual_flags, TX_WAIT_FOREVER);
 if ((actual_flags & CDCACM_ACTIVATE_FLAG) && (TX_SUCCESS == status))
    {
 /* do nothing */
    }
 else if (!(actual_flags & CDCACM_ACTIVATE_FLAG))
    {
 /* do nothing */
    }
 /* usb pcdc operations will echo the user input on serial terminal*/
 while (true)
    {
       state = ux_device_class_dfu_state_get(dfu);
```

```
 if (UX_SYSTEM_DFU_STATE_APP_IDLE == state)
      {
            usbx_pcdc_operations();
      }
       tx_thread_sleep(1);
       tx_event_flags_get(&g_cdcacm_event_flags0, CDCACM_FLAG, TX_OR, &actual_flags,
TX_WAIT_FOREVER);
 if (actual_flags & CDCACM_FLAG)
      {
 if (actual_flags & DFU_DETACH_REQUEST_FLAG)
      {
         state = ux_device_class_dfu_state_get(dfu);
 if (UX_SYSTEM_DFU_STATE_APP_DETACH == state)
      {
                  mode_change_cdc_to_dfu();
      }
 else if (UX_SYSTEM_DFU_STATE_DFU_MANIFEST_WAIT_RESET == state)
      {
                  mode_change_dfu_to_cdc();
      }
               tx_event_flags_set(&g_cdcacm_event_flags0,
~(DFU_DETACH_REQUEST_FLAG), TX_AND);
      }
      }
       tx_thread_sleep(100);
    }
}
/*****************************************************************************
 * Function Name : usbx_pcdc_operations
 * Description : In this function, it performs the usb write/read operation and echo
back the user input on serial terminal
 * Arguments : none
 * Return value : none
 *****************************************************************************/
```

```
static void usbx_pcdc_operations (void)
{
    UX_SLAVE_CLASS_DFU       * dfu;
    UCHAR                      state;
    UINT                      status    = UX_SUCCESS;
    uint32_t                   data_size = RESET_VALUE;
 volatile UX_SLAVE_DEVICE * device;
    device = &_ux_system_slave->ux_system_slave_device;
 /* Wait until usb device is configured to slave */
 if (device->ux_slave_device_state != UX_DEVICE_CONFIGURED)
    {
 return;
    }
 /* Clear the buffer */
    memset(g_buf, RESET_VALUE, sizeof(g_buf));
 /* USB Reads the input data from the user from serial terminal */
    status = ux_device_class_cdc_acm_read(g_cdc, g_buf, DATA_LEN, &g_actual_length);
 /* update the data length from the read input */
    data_size = g_actual_length;
    state = ux_device_class_dfu_state_get(dfu);
 if (UX_SYSTEM_DFU_STATE_APP_DETACH == state)
    {
 return;
    }
 /* Write back the read data on to the serial terminal */
    status = ux_device_class_cdc_acm_write(g_cdc, g_buf, data_size,
&g_actual_length);
 if (g_actual_length == device->ux_slave_device_descriptor.bMaxPacketSize0)
    {
 /* 0-Length-Packet */
       ux_device_class_cdc_acm_write(g_cdc, g_buf, 0, &g_actual_length);
    }
}
/**********************************************************************************
```

```
 * End of function usbx_pcdc_operations
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : ux_cdc_device0_instance_activate
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
    tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_ACTIVATE_FLAG, TX_OR);
}
/***********************************************************************************
 * End of function ux_cdc_device0_instance_activate
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 *
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
    tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_DEACTIVATE_FLAG, TX_OR);
}
/***********************************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 ***********************************************************************************/
/***********************************************************************************
```

```
 * Function Name : ux_dfu_device0_instance_activate
 * Description : Get instance
 * Arguments : void * dfu_instance : Pointer to the area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_dfu_device0_instance_activate (void * dfu_instance)
{
 /* Save the DFU instance. */
    g_dfu = (UX_SLAVE_CLASS_DFU *) dfu_instance;
    tx_event_flags_set(&g_dfu_event_flags0, DFU_ACTIVATE_FLAG, TX_OR);
}
/***********************************************************************************
 * End of function ux_dfu_device0_instance_activate
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : ux_dfu_device0_instance_deactivate
 * Description : Clear dfu_instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 ***********************************************************************************/
static void ux_dfu_device0_instance_deactivate (void * dfu_instance)
{
 FSP_PARAMETER_NOT_USED(dfu_instance);
    g_dfu = UX_NULL;
    tx_event_flags_set(&g_dfu_event_flags0, DFU_DEACTIVATE_FLAG, TX_OR);
}
/***********************************************************************************
 * End of function ux_dfu_device0_instance_deactivate
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : usbx_status_callback
 * Description : Callback on device state change
 * Arguments : ULONG status : New USB Device Status
 * Return value : 0
```

```
 ***********************************************************************/
UINT usbx_status_callback (ULONG status)
{
    UX_SLAVE_CLASS_DFU * dfu;
    UCHAR                state;
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
     {
           tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_FLAG, TX_OR);
 break;
     }
 case UX_DEVICE_REMOVED:
     {
           tx_event_flags_set(&g_cdcacm_event_flags0, ~CDCACM_FLAG, TX_AND);
 break;
     }
 case UX_DEVICE_FORCE_DISCONNECT:
     {
        state = ux_device_class_dfu_state_get(dfu);
 if ((UX_SYSTEM_DFU_STATE_APP_DETACH == state) ||
               (UX_SYSTEM_DFU_STATE_DFU_MANIFEST_WAIT_RESET == state))
     {
               tx_event_flags_set(&g_cdcacm_event_flags0, DFU_DETACH_REQUEST_FLAG,
TX_OR);
     }
 break;
     }
 default:
     {
 /* do nothing */
 break;
     }
   }
```

```
 return 0;
}
/**************************************************************************
 * Function Name : dfu_register_function
 * Description : Register the DFU class device
 * Arguments : UINT if_num : Interface number
 * Return value : none
 **************************************************************************/
static void dfu_register_function (UINT if_num)
{
    UINT status = UX_SUCCESS;

g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_instance_activate   =
ux_dfu_device0_instance_activate;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_instance_deactivate
=
       ux_dfu_device0_instance_deactivate;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_framework =
       g_dfu_device_framework_full_speed;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_framework_length =
       DFU_DEVICE_FRAME_LENGTH_FULL_SPEED;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_write      =
dfu_dammy_write;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_get_status =
dfu_dammy_get_status;
    g_ux_device_class_dfu_parameter.ux_slave_class_dfu_parameter_notify    =
dfu_dammy_notify;
    status =
       ux_device_stack_class_register(_ux_system_slave_class_dfu_name,
                                      ux_device_class_dfu_entry,
                                      CONFIG_NUMB1,
                                      if_num,
                                      (void *) &g_ux_device_class_dfu_parameter);
 if (UX_SUCCESS != status)
```

```
    {
 while (1)
      {
         tx_thread_sleep(1);
      }
    }
}
/**************************************************************************
 * End of function dfu_register_function
 **************************************************************************/
/**************************************************************************
 * Function Name : dfu_dammy_write
 * Description : Write firmware data to media (e.g. non-volatile memory)
 * Arguments : VOID *dfu, ULONG block_number, UCHAR * data_pointer, ULONG length,
ULONG *media_status
 * Return value : UX_SUCCESS (or write err result)
 **************************************************************************/
static UINT dfu_dammy_write (VOID * dfu, ULONG block_number, UCHAR * data_pointer,
ULONG length, ULONG * media_status)
{
 return UX_SUCCESS;
}
/**************************************************************************
 * End of function dfu_dammy_write
 **************************************************************************/
/**************************************************************************
 * Function Name : dfu_dammy_get_status
 * Description : Outputs the status of writing firmware data to media (e.g. non-
volatile memory)
 * Arguments : VOID *dfu, ULONG *media_status
 * Return value : UX_SUCCESS
 **************************************************************************/
static UINT dfu_dammy_get_status (VOID * dfu, ULONG * media_status)
{
```

```
    *media_status = UX_SLAVE_CLASS_DFU_MEDIA_STATUS_OK;
 return UX_SUCCESS;
}
/**************************************************************************
 * End of function dfu_dammy_get_status
 **************************************************************************/
/**************************************************************************
 * Function Name : dfu_dammy_notify
 * Description : Notifications about transferring firmware data to applications
 * Arguments : VOID *dfu, ULONG notification
 * Return value : UX_SUCCESS
 **************************************************************************/
static UINT dfu_dammy_notify (VOID * dfu, ULONG notification)
{
 return UX_SUCCESS;
}
/**************************************************************************
 * End of function dfu_dammy_notify
 **************************************************************************/
/**************************************************************************
 * Function Name : mode_change_cdc_to_dfu
 * Description : Switch from normal mode (CDC+DFU) to DFU mode.
 * Arguments : none
 * Return value : none
 **************************************************************************/
static void mode_change_cdc_to_dfu (void)
{
    INT err;
    UX_SLAVE_CLASS_DFU * dfu;
    UCHAR                state;
    UINT                status;
 if (DFU_FIRM_UPDATE_MAX_TRY <= g_dfu_firmware_update_done_count)
    {
 return;
```

```
    }
    err = R_USB_Close(&g_basic1_ctrl);
 if (FSP_SUCCESS != err)
    {
 while (1)
      {
            ;
      }
    }
    status = ux_device_stack_class_unregister(_ux_system_slave_class_cdc_acm_name,
_ux_device_class_cdc_acm_entry);
    status = ux_device_stack_class_unregister(_ux_system_slave_class_dfu_name,
_ux_device_class_dfu_entry);
    status = ux_device_stack_uninitialize();
    status = ux_device_stack_initialize(UX_NULL,
                                        UX_NULL,
                                        g_dfu_device_framework_full_speed,
                                        DFU_DEVICE_FRAME_LENGTH_FULL_SPEED,
                                        g_dfu_string_framework,
                                        DFU_STRING_FRAMEWORK_LENGTH,
                                  g_language_id_framework,
                                  LANGUAGE_ID_FRAME_WORK_LENGTH,
                                        &usbx_status_callback);
    dfu_register_function(INTERFACE_NUMB0); /* Input : IF number */
    err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
 while (1)
      {
            ;
      }
    }
}
/***********************************************************************************
```

```
 * End of function mode_change_cdc_to_dfu
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : mode_change_dfu_to_cdc
 * Description : Switches from DFU mode to normal mode (CDC+DFU).
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
static void mode_change_dfu_to_cdc (void)
{
    UX_SLAVE_CLASS_DFU * dfu;
    UCHAR                state;
    UINT                status;
    INT err;
 if (DFU_FIRM_UPDATE_MAX_TRY <= g_dfu_firmware_update_done_count)
    {
 return;
    }
    g_dfu_firmware_update_done_count++;
    err = R_USB_Close(&g_basic0_ctrl);
 if (FSP_SUCCESS != err)
    {
 while (1)
     {
           ;
     }
    }
    status = ux_device_stack_class_unregister(_ux_system_slave_class_dfu_name,
_ux_device_class_dfu_entry);
    status = ux_device_stack_uninitialize();
    status = ux_device_stack_initialize(g_device_framework_hi_speed,
                                DEVICE_FRAME_LENGTH_HIGH_SPEED,
                                g_device_framework_full_speed,
                                DEVICE_FRAME_LENGTH_FULL_SPEED,
```

```
                                g_string_framework,
                                STRING_FRAMEWORK_LENGTH,
                                g_language_id_framework,
                                LANGUAGE_ID_FRAME_WORK_LENGTH,
                                      &usbx_status_callback);
    status = ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                   _ux_device_class_cdc_acm_entry,
                                         CONFIG_NUMB1,
                                   INTERFACE_NUMB0,
                                   (void *) &g_ux_device_class_cdc_acm0_parameter);
    dfu_register_function(INTERFACE_NUMB2); /* Input:IF number */
    err = R_USB_Open(&g_basic1_ctrl, &g_basic1_cfg);
 if (FSP_SUCCESS != err)
    {
 while (1)
     {
           ;
     }
    }
}
```

### USBX Composite Example

USBX Composite (PCDC + PMSC) example is as follows.

```
/* Mempool size of 14k is required for USBX device class pre built libraries
 * and it is valid only if it with default USBX configurations. */
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / BYTE_SIZE];
static ULONG    actual_flags = RESET_VALUE;
/* Mempool size of 18k is required for USBX device class pre built libraries
 * and it is valid only if it with default USBX configurations. */
static uint32_t g_ux_pool_memory[MEMPOOL_SIZE / BYTE_SIZE];
static UX_SLAVE_CLASS_CDC_ACM_PARAMETER g_ux_device_class_cdc_acm0_parameter;
static UX_SLAVE_CLASS_CDC_ACM * g_cdc;
static ULONG g_actual_length;
```

```
static uint8_t g_buf[DATA_LEN];
static bool b_print_status = false;
/* Private function declarations. */
UINT usbx_status_callback(ULONG status);
static void ux_cdc_device0_instance_activate(void * cdc_instance);
static void ux_cdc_device0_instance_deactivate(void * cdc_instance);
static void usb_connection_status_check(void);
static void usbx_pcdc_operations(void);
/***********************************************************************
 * Function Name : usbx_composite_pcdc_mmsc_sample
 * Description : Application Thread entry function
 * Arguments : none
 * Return value : none
 ***********************************************************************/
void usbx_composite_pcdc_mmsc_sample (void)
{
 /* ux_system_initialization */
    ux_system_initialize(g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, RESET_VALUE);
 /* ux_device stack initialization */
    ux_device_stack_initialize(g_device_framework_hi_speed,
                               DEVICE_FRAME_LENGTH_HIGH_SPEED,
                               g_device_framework_full_speed,
                               DEVICE_FRAME_LENGTH_FULL_SPEED,
                               g_string_framework,
                               STRING_FRAMEWORK_LENGTH,
                               g_language_id_framework,
                               LANGUAGE_ID_FRAME_WORK_LENGTH,
                               &usbx_status_callback);
 /* The activate command is used when the host has sent a SET_CONFIGURATION command
  * and this interface has to be mounted. Both Bulk endpoints have to be mounted
  * and the cdc_acm thread needs to be activated. */
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate =
ux_cdc_device0_instance_activate;
 /* The deactivate command is used when the device has been extracted.
```

```
 * The device endpoints have to be dismounted and the cdc_acm thread canceled. */
   g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
      ux_cdc_device0_instance_deactivate;
/* ux_device stack class registration */
   ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                  _ux_device_class_cdc_acm_entry,
                                  CONFIG_NUMB,
                                  INTERFACE_NUMB0,
                                  (void *) &g_ux_device_class_cdc_acm0_parameter);
/* Open usb driver */
R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
/* Wait until device inserted.*/
   tx_event_flags_get(&g_msc_event_flags0, USB_MSC_PLUG_IN, TX_AND_CLEAR,
&actual_flags, TX_WAIT_FOREVER);
if (USB_MSC_PLUG_IN == actual_flags)
   {
      ;                             /* USB MSC device is plugged in */
   }
/* Reset the event flag */
   actual_flags = RESET_VALUE;
while (true)
   {
/* Check if USB is plugged out.*/
      tx_event_flags_get(&g_msc_event_flags0, USB_MSC_PLUG_OUT, TX_AND_CLEAR,
&actual_flags, TX_NO_WAIT);
if (USB_MSC_PLUG_OUT == (actual_flags & USB_MSC_PLUG_OUT))
     {
/* Reset the event flag */
     actual_flags = RESET_VALUE;
     }
/* Check if USB is plugged in. */
      tx_event_flags_get(&g_msc_event_flags0, USB_MSC_PLUG_IN, TX_AND_CLEAR,
&actual_flags, TX_NO_WAIT);
if (USB_MSC_PLUG_IN == (actual_flags & USB_MSC_PLUG_IN))
```

```
      {
 /* Reset the event flag */
      actual_flags = RESET_VALUE;
      }
       usbx_pcdc_operations();
       tx_thread_sleep(1);
    }
}
/**********************************************************************************
 * End of function usbx_composite_pcdc_mmsc_sample
 *********************************************************************************/
/**********************************************************************************
 * Function Name : usbx_status_callback
 * Description : In this function, usb callback events will be captured into one
variable.
 * Arguments : ULONG status : USB status. Whenever any event occurred, status gets
update.
 * Return value : UX_SUCCESS
 *********************************************************************************/
UINT usbx_status_callback (ULONG status)
{
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
      {
 /* Set USB PLUG-IN event.*/
            tx_event_flags_set(&g_msc_event_flags0, USB_MSC_PLUG_IN, TX_OR);
            tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_FLAG, TX_OR);
 break;
      }
 case UX_DEVICE_REMOVED:
      {
 /* Set USB PLUG-OUT event.*/
            tx_event_flags_set(&g_msc_event_flags0, USB_MSC_PLUG_OUT, TX_OR);
```

```
            tx_event_flags_set(&g_cdcacm_event_flags0, ~CDCACM_FLAG, TX_AND);
 break;
     }
 default:
     {
 /* do nothing */
 break;
     }
   }
 return 0;
}
/**********************************************************************************
 * End of function usbx_status_callback
 *********************************************************************************/
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_activate
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 *********************************************************************************/
static void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
    tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_ACTIVATE_FLAG, TX_OR);
}
/**********************************************************************************
 * End of function ux_cdc_device0_instance_activate
 *********************************************************************************/
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
```

```
 ***********************************************************************/
static void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
    tx_event_flags_set(&g_cdcacm_event_flags0, CDCACM_DEACTIVATE_FLAG, TX_OR);
}
/***********************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 ***********************************************************************/
/***********************************************************************
 * Function Name : usb_connection_status_check
 * Description : In this function, checks the USB device status
 * and notifies the user by printing the status message
 * Arguments : none
 * Return value : none
 ***********************************************************************/
static void usb_connection_status_check (void)
{
    ULONG actual_flags = RESET_VALUE;
 /* wait for usb connection event */
    tx_event_flags_get(&g_cdcacm_event_flags0, CDCACM_FLAG, TX_OR, &actual_flags,
TX_WAIT_FOREVER);
 if (actual_flags & CDCACM_FLAG)
    {
 if (true != b_print_status)
     {
           b_print_status = true;    // flag is updated
     }
 else
     {
 /* do nothing */
     }
    }
```

```
 else
    {
       b_print_status = false;       // clear the flag
 while (!(actual_flags & CDCACM_FLAG))
     {
           ; /* wait until the event update */
     }
    }
}
/***********************************************************************************
 * End of function usb_connection_status_check
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : usbx_pcdc_operations
 * Description : In this function, it performs the usb write/read operation
 * and echo back the user input on serial terminal
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
static void usbx_pcdc_operations (void)
{
    uint32_t data_size = RESET_VALUE;
 volatile UX_SLAVE_DEVICE * device;
    device = &_ux_system_slave->ux_system_slave_device;
 /* Verify the status of usb */
    usb_connection_status_check();
    tx_event_flags_get(&g_cdcacm_event_flags0, CDCACM_ACTIVATE_FLAG, TX_OR,
&actual_flags, TX_WAIT_FOREVER);
 /* USB writes the display message on serial terminal */
    ux_device_class_cdc_acm_write(g_cdc,
                                  (UCHAR *) "\r\nEnter any Key to echo back the
entered data \r\nUser Input :",
                                  WRITE_DATA_LEN,
                                  &g_actual_length);
```

```
 /* Clear the buffer */
    memset(g_buf, RESET_VALUE, sizeof(g_buf));
 /* USB Reads the input data from the user from serial terminal */
    ux_device_class_cdc_acm_read(g_cdc, g_buf, DATA_LEN, &g_actual_length);
 /* update the data length from the read input */
    data_size = g_actual_length;
 /* Write back the read data on to the serial terminal */
    ux_device_class_cdc_acm_write(g_cdc, g_buf, data_size, &g_actual_length);
}
/******************************************************************************
 * End of function usbx_pcdc_operations
 ******************************************************************************/
```

### USBX OTG Example

OTG example is as follows.

```
#define VALUE_108 (108)
#define VALUE_105 (105)
#define VALUE_98 (98)
#define VALUE_2 (2)
static volatile ULONG g_apl_status_peri = 0;
volatile uint8_t g_apl_state = UX_DEVICE_REMOVED;
static volatile ULONG g_apl_usb_mode = UX_OTG_MODE_IDLE;
static volatile ULONG g_change_device_mode = UX_OTG_MODE_IDLE;
static UX_HOST_CLASS_CDC_ACM * g_p_cdc_acm_host = UX_NULL;
static UX_SLAVE_CLASS_CDC_ACM * volatile g_p_cdc_peri = UX_NULL;
/******************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 ******************************************************************************/
UINT apl_status_change_cb (ULONG status)
{
```

```
 // Debug OTG-A Detach
 if ((UX_DEVICE_REMOVED == g_apl_status_peri) && (UX_DEVICE_RESUMED == status))
    {
 return UX_SUCCESS;
    }
    g_apl_status_peri = status;
 return UX_SUCCESS;
}
/**********************************************************************************
 * End of function apl_status_change_cb
 *********************************************************************************/
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_activate
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 *********************************************************************************/
void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_p_cdc_peri = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
}
/**********************************************************************************
 * End of function ux_cdc_device0_instance_activate
 *********************************************************************************/
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
 *********************************************************************************/
void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
```

```
    g_p_cdc_peri = UX_NULL;
}
/* USB OTG Application */
/**********************************************************************************
 * Function Name : apl_device_swich_complete_cb
 * Description : Callback function called when switcning Host or Peri
 * Arguments : UX_OTG_MODE_SLAVE/UX_OTG_MODE_HOST/UX_OTG_MODE_IDLE
 * Return value : none
 *********************************************************************************/
void apl_device_swich_complete_cb (ULONG mode)
{
 if (UX_OTG_MODE_SLAVE == mode)
    {
       _ux_system_otg->ux_system_otg_slave_role_swap_flag = 0;
    }
    g_change_device_mode = mode;
}
/**********************************************************************************
 * Function Name : otg_host_apl
 * Description : OTG sample program
 * Arguments : none
 * Return value : none
 *********************************************************************************/
void apl_otg_sample (void)
{
    uint8_t   is_host_request_flag = USB_NO;
    uint8_t   is_host_apl_complete = USB_NO;
 fsp_err_t err;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
 // B device initialization
    ux_device_stack_initialize(UX_NULL,
                               UX_NULL,
                               g_device_framework_full_speed,
                               VALUE_98,
```

```
                                g_string_framework,
                                VALUE_105,
                                g_language_id_framework,
                                VALUE_2,
                                apl_status_change_cb);
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate   =
ux_cdc_device0_instance_activate;
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
       ux_cdc_device0_instance_deactivate;
    ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                   _ux_device_class_cdc_acm_entry,
                                   1,
                                   0x00,
                                   (void *) &g_ux_device_class_cdc_acm0_parameter);
 // A device initialization
    ux_host_stack_initialize(ux_host_usr_event_notification);
 R_USB_OtgCallbackSet(&g_basic0_ctrl, apl_device_swich_complete_cb);
#if defined(APL_USB_OTG_A_DEVICE)
    err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl, &g_external_irq0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl);
    assert(FSP_SUCCESS == err);
#endif
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
 while (1)
     {
           ;
     }
    }
#if defined(APL_USB_OTG_A_DEVICE)
 while (1)
    {
```

```
 if (UX_OTG_MODE_HOST == g_change_device_mode)
      {
 break;
      }
    }
#endif  /* defined(APL_USB_OTG_A_DEVICE) */
 while (1)
    {
 if (g_change_device_mode != g_apl_usb_mode)
      {
            g_apl_usb_mode = g_change_device_mode;
 switch (g_apl_usb_mode)
      {
 case UX_OTG_MODE_HOST:
      {
                is_host_request_flag = USB_NO;
 if (USB_NO == is_host_apl_complete)
      {
                    otg_host_apl();
                    is_host_apl_complete = USB_YES;
      }
 break;
      }
 case UX_OTG_MODE_SLAVE:
      {
                is_host_apl_complete = USB_NO;
 if (USB_NO == is_host_request_flag)
      {
                    err = otg_peri_apl();
 if (0 == err)
      {
 if (UX_OTG_FEATURE_A_HNP_SUPPORT ==
                            (_ux_system_otg->ux_system_otg_slave_set_feature_flag
& UX_OTG_FEATURE_A_HNP_SUPPORT))
```

```
     {
                              _ux_system_otg->ux_system_otg_slave_role_swap_flag =
UX_OTG_HOST_REQUEST_FLAG;
                              is_host_request_flag = USB_YES;
     }
     }
     }
 break;
     }
 default:
     {
 /* UX_MODE_IDLE */
                  is_host_request_flag = USB_NO;
                  is_host_apl_complete = USB_NO;
 break;
     }
     }
     }
    }
}
/**************************************************************************
 * End of function apl_otg_sample
 **************************************************************************/
/* USB Host Application */
/**************************************************************************
 * Function Name : ux_host_usr_event_notification
 * Description : Callback function called when completing USB event
 * Arguments : ULONG event : Completed USB Event
 * : UX_HOST_CLASS *host_class : Pointer to UX_HOST_CLASS structure
 * : VOID * instance : Pointer to HCDC instance
 * Return value : UX_SUCCESS
 **************************************************************************/
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
```

```
{
    (void) host_class;
 if (UX_DEVICE_INSERTION == event) /* Check if there is a device insertion. */
    {
       g_p_cdc_host = (UX_HOST_CLASS_CDC_ACM *) instance;
 if (UX_HOST_CLASS_CDC_DATA_CLASS !=
            g_p_cdc_host->ux_host_class_cdc_acm_interface->ux_interface_descriptor.bI
nterfaceClass)
     {
 /* It seems the DATA class is on the second interface. Or we hope ! */
            g_p_cdc_host = g_p_cdc_host->ux_host_class_cdc_acm_next_instance;
 /* Check again this interface, if this is not the data interface, we give up. */
 if (UX_HOST_CLASS_CDC_DATA_CLASS !=
              g_p_cdc_host->ux_host_class_cdc_acm_interface->ux_interface_descriptor.
bInterfaceClass)
     {
 /* We did not find a proper data interface. */
               g_p_cdc_host = UX_NULL;
     }
     }
 if (UX_NULL != g_p_cdc_host)
     {
            g_host_apl_event = UX_DEVICE_INSERTION;
     }
       tx_thread_wait_abort(&new_thread0);
    }
 else if ((UX_DEVICE_REMOVAL == event) || (UX_DEVICE_DISCONNECTION == event)) /*
Check if there is a device removal. */
    {
       g_host_apl_event = UX_DEVICE_REMOVAL;
       g_p_cdc_host    = UX_NULL;
    }
 return UX_SUCCESS;
}
```

```
/**************************************************************************
 * End of function ux_host_usr_event_notification
 **************************************************************************/
/**************************************************************************
 * Function Name : otg_host_apl
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 **************************************************************************/
void otg_host_apl (void)
{
    ULONG status;
    m
 for (counter = 0; counter < DATA_LEN; counter++)
    {
       g_write_buf_host[counter] = (uint8_t) counter;
    }
    g_host_apl_event = 0;
 while (1)
    {
 if (UX_DEVICE_INSERTION == g_host_apl_event)
     {
     g_host_apl_event = 0;
 for (countor = 0; countor < 5000; countor++)
     {
 if (0 == g_is_communicate)
     {
        tx_thread_sleep(100);
                g_is_communicate = 1;
     }
              g_write_actual_length = 0;
              status               = ux_host_class_cdc_acm_write(g_p_cdc_host,
                                                                 g_write_buf_host,
                                                                 DATA_LEN,
```

```
&g_write_actual_length_host);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_write_actual_length_host))
      {
                  g_read_actual_length_host = 0;
                  buffer_clear(g_host_read_buf);
      status = ux_host_class_cdc_acm_read(g_p_cdc_host,
                                                          g_read_buf_host,
                                                          DATA_LEN,
                                                          &g_read_actual_length_host);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_read_actual_length_host))
      {
 for (counter = 0; counter < DATA_LEN; counter++)
      {
 if ((uint8_t) counter != g_read_buf_host[counter])
      {
 while (1)
      {
                    ;
      }
      }
      }
      }
 else
      {
 break;
      }
      }
 else
      {
 break;
      }
      }
      }
```

```
 else if (UX_DEVICE_REMOVAL == g_host_apl_event)
      {
      g_host_apl_event = 0;
 break;
      }
    }
}
/**********************************************************************************
 * End of function otg_host_apl
 *********************************************************************************/
/* USB Peripheral Application */
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_activate
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
 *********************************************************************************/
void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_p_cdc_peri = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
}
/**********************************************************************************
 * End of function ux_cdc_device0_instance_activate
 *********************************************************************************/
/**********************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
 *********************************************************************************/
void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
```

```
    g_p_cdc_peri = UX_NULL;
}
/**********************************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 *********************************************************************************/
/**********************************************************************************
 * Function Name : otg_peri_apl
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *********************************************************************************/
uint8_t otg_peri_apl (void)
{
    UINT    ret;
    ULONG    size;
    uint16_t counter = 0;
    g_apl_status_peri = 0;
 for (counter = 0; counter < 5000; )
    {
 if (UX_DEVICE_CONFIGURED == g_apl_status_peri)
     {
 while (g_p_cdc_peri == UX_NULL)
     {
             ;
     }
           ret = _ux_device_class_cdc_acm_read(g_p_cdc_peri, g_buf_peri, DATA_LEN,
&g_actual_length_peri);
 if (UX_SUCCESS == ret)
     {
               size = g_actual_length_peri;
               ux_device_class_cdc_acm_write(g_p_cdc_peri, g_buf_peri, size,
&g_actual_length_peri);
               counter++;
     }
```

```
     }
 else if (UX_DEVICE_REMOVED == g_apl_status_peri)
     {
 if (0 != counter)
     {
 break;
     }
     }
   }
 if (5000 == counter)
   {
 return APL_SUCCESS;
   }
 return APL_ERROR;
}
/**********************************************************************************
 * End of function otg_peri_apl
 *********************************************************************************/
```

### 5.2.6.2 CAN (r_can)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_CAN_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg) |
| fsp_err_t | R_CAN_Close (can_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_CAN_Write (can_ctrl_t *const p_api_ctrl, uint32_t mailbox, can_frame_t *const p_frame) |
| fsp_err_t | R_CAN_Read (can_ctrl_t *const p_api_ctrl, uint32_t mailbox, can_frame_t *const p_frame) |
| fsp_err_t | R_CAN_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode) |
| fsp_err_t | R_CAN_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info) |

| | |
|---|---|
| fsp_err_t | R_CAN_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface.

# Overview

The Controller Area network (CAN) HAL module provides a high-level API for CAN applications and supports the CAN peripherals available on RA microcontroller hardware. A user-callback function must be defined that the driver will invoke when transmit, receive or error interrupts are received. The callback is passed a parameter which indicates the channel, mailbox and event as well as the received data (if available).

### Features

- Supports both standard (11-bit) and extended (29-bit) messaging formats
- Supports speeds upto 1 Mbps
- Support for bit timing configuration as defined in the CAN specification
- Supports up to 32 transmit or receive mailboxes with standard or extended ID frames
- Optional support for a 4-stage transmit and receive FIFO
- Receive mailboxes can be configured to capture either data or remote CAN Frames
- Receive mailboxes can be configured to receive a range of IDs using mailbox masks
- Mailboxes can be configured with Overwrite or Overrun mode
- Supports a user-callback function when transmit, receive, or error interrupts are received

# Configuration

### Build Time Configurations for r_can

The following build time configurations are defined in fsp_cfg/r_can_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| FIFO Support | • Disabled<br>• Enabled | Disabled | When FIFOs are enabled, a transmit FIFO replaces mailboxes 24-27 and a receive FIFO replaces mailboxes 28-31. |

### Configurations for Connectivity > CAN (r_can)

This module can be added to the Stacks tab via New Stack > Connectivity > CAN (r_can). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_can0 | Module name. |
| Channel | Channel should be 0 or 1 | 0 | Specify the CAN channel to use. |
| Clock Source | MCU Specific Options | | Select the CAN clock source. |
| Overwrite/Overrrun Mode | • Overwrite Mode<br>• Overrrun Mode | Overwrite Mode | Select whether receive mailbox will be overwritten or overrun if data is not read in time. |
| Global ID Mode | • Standard ID Mode<br>• Extended ID Mode<br>• Mixed ID Mode | Standard ID Mode | Select whether the driver will use CAN Standard IDs, Extended IDs or a mix of both. |
| Number of Mailboxes | • 4 Mailboxes<br>• 8 Mailboxes<br>• 16 Mailboxes<br>• 24 Mailboxes<br>• 32 Mailboxes | 32 Mailboxes | Select 4, 8, 16, 24 or 32 mailboxes. In FIFO mailbox mode up to 24 mailboxes are available. |
| Baud Rate Settings | | | |
| Baud Rate Settings > Auto-generated Settings | | | |
| Sample-Point (%) | Must be a valid integer between 0 and 100. Ignore when Override Baud Settings is Enabled. | 75 | Sample-Point = (TSEG1 + 1) / (TSEG1 + TSEG2 + 1). |
| CAN Baud Rate (Hz) | Must be a valid integer configurable upto maximum 1MHz. Ignore when Override Baud Settings is Enabled. | 500000 | Specify baud rate in Hz.<br><br>If the requested baud rate cannot be achieved, the settings with the largest possible baud rate that is less than or equal to the requested baud rate is used. If multiple combinations would result in the best baud rate, the combination with the least absolute error for the ratio is chosen. The theoretical calculated baud rate |

| | | | |
|---|---|---|---|
| | | | and ratio are printed in a comment in the generated can_bit_timing_cfg_t structure. |
| Baud Rate Settings > Override Auto-generated Settings | | | |
| Override Baud Settings | • Enabled<br>• Disabled | Disabled | Override calculated baudrate parameters and instead use the ones specified below. This option ignores the parameters specified under Sample-Point (%) and CAN Baud Rate (Hz) |
| Baud Rate Prescaler | Value must be a non-negative integer between 1 and 1024. | 1 | Specify division value of baud rate prescaler (baud rate prescalar + 1). |
| Time Segment 1 | Refer to the RA Configuration tool for available options. | 4 Time Quanta | Select the time segment 1 value. (4-16). Check module usage notes for how to calculate this value. |
| Time Segment 2 | • 2 Time Quanta<br>• 3 Time Quanta<br>• 4 Time Quanta<br>• 5 Time Quanta<br>• 6 Time Quanta<br>• 7 Time Quanta<br>• 8 Time Quanta | 2 Time Quanta | Select the time segment 2 value (2-8). Check module usage notes for how to calculate this value. |
| Synchronization Jump Width | • 1 Time Quanta<br>• 2 Time Quanta<br>• 3 Time Quanta<br>• 4 Time Quanta | 1 Time Quanta | Select the Synchronization Jump Width value (1-4). Check module usage notes for how to calculate this value. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | can_callback | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs. |
| Interrupt Priority Level | MCU Specific Options | | Transmit/Receive/Error interrupt priority. |
| Transmit FIFO Interrupt Mode | • Every Message<br>• Empty | Every Message | Select whether the receive FIFO should |

| | | | |
|---|---|---|---|
| | | | throw an interrupt on every received message or when it becomes empty. |
| Input | | | |
| Input > Receive FIFO | | | |
| Input > Receive FIFO > Receive ID 1 | | | |
| ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0 | Select the first ID for the receive FIFO, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. |
| ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the receive FIFO is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Frame Type | • Data Mailbox<br>• Remote Mailbox | Remote Mailbox | Select whether the receive FIFO is used to capture data frames or remote frames. |
| Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for the receive FIFO. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |
| Input > Receive FIFO > Receive ID 2 | | | |
| ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0 | Select the second ID for the receive FIFO, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. |
| ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the receive FIFO is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |

| | | | |
|---|---|---|---|
| Frame Type | • Data Mailbox<br>• Remote Mailbox | Remote Mailbox | Select whether the receive FIFO is used to capture data frames or remote frames. |
| Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for the receive FIFO. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |

Input > Mailbox 0-3 Group

Input > Mailbox 0-3 Group > Mailbox ID

| | | | |
|---|---|---|---|
| Mailbox 0 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0 | Select the receive ID for mailbox 0, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 1 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 1 | Select the receive ID for mailbox 1, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 2 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 2 | Select the receive ID for mailbox 2, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 3 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 3 | Select the receive ID for mailbox 3, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |

Input > Mailbox 0-3 Group > Mailbox ID Mode

| | | | |
|---|---|---|---|
| Mailbox 0 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 1 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 2 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 3 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 0-3 Group > Mailbox Type | | | |
| Mailbox 0 Type | • Receive Mailbox<br>• Transmit Mailbox | Transmit Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 1 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 2 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 3 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |

Input > Mailbox 0-3 Group > Mailbox Frame Type

| | | | |
|---|---|---|---|
| Mailbox 0 Frame Type | • Data Mailbox<br>• Remote Mailbox | Remote Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 1 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 2 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 3 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 0-3 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 0-3. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |

Input > Mailbox 4-7 Group

Input > Mailbox 4-7 Group > Mailbox ID

| | | | |
|---|---|---|---|
| Mailbox 4 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 4 | Select the receive ID for mailbox 4, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 5 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 5 | Select the receive ID for mailbox 5, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used |

| | | | |
|---|---|---|---|
| | | | when the mailbox is set as transmit type. |
| Mailbox 6 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 6 | Select the receive ID for mailbox 6, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 7 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 7 | Select the receive ID for mailbox 7, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 4-7 Group > Mailbox ID Mode | | | |
| Mailbox 4 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 5 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 6 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 7 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID |

| | | | |
|---|---|---|---|
| | | | Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 4-7 Group > Mailbox Type | | | |
| Mailbox 4 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 5 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 6 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 7 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 4-7 Group > Mailbox Frame Type | | | |
| Mailbox 4 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 5 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 6 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 7 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 4-7 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | >Select the Mask for mailboxes 4-7. In Mixed ID Mode the Standard ID mask is the upper |

| | | | |
|---|---|---|---|
| | | | 11 bits of the full 29-bit mask value. |
| Input > Mailbox 8-11 Group | | | |
| Input > Mailbox 8-11 Group > Mailbox ID | | | |
| Mailbox 8 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 8 | Select the receive ID for mailbox 8, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 9 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 9 | Select the receive ID for mailbox 9, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 10 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 10 | Select the receive ID for mailbox 10, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 11 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 11 | Select the receive ID for mailbox 11, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 8-11 Group > Mailbox ID Mode | | | |
| Mailbox 8 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed |

| | | | |
|---|---|---|---|
| | | | ID Mode'. |
| Mailbox 9 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 10 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 11 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 8-11 Group > Mailbox Type | | | |
| Mailbox 8 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 9 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 10 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 11 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 8-11 Group > Mailbox Frame Type | | | |
| Mailbox 8 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |

| | | | |
|---|---|---|---|
| Mailbox 9 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 10 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 11 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 8-11 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 8-11. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |

Input > Mailbox 12-15 Group

Input > Mailbox 12-15 Group > Mailbox ID

| | | | |
|---|---|---|---|
| Mailbox 12 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 12 | Select the receive ID for mailbox 12, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 13 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 13 | Select the receive ID for mailbox 13, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 14 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 14 | Select the receive ID for mailbox 14, between 0 and 0x7ff when using standard |

| | | | |
|---|---|---|---|
| | | | IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 15 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 15 | Select the receive ID for mailbox 15, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 12-15 Group > Mailbox ID Mode | | | |
| Mailbox 12 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 13 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 14 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 15 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 12-15 Group > Mailbox Type | | | |

| | | | |
|---|---|---|---|
| Mailbox 12 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 13 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 14 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 15 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 12-15 Group > Mailbox Frame Type | | | |
| Mailbox 12 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 13 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 14 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 15 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 12-15 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 12-15. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |
| Input > Mailbox 16-19 Group | | | |

Input > Mailbox 16-19 Group > Mailbox ID

| | | | |
|---|---|---|---|
| Mailbox 16 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 16 | Select the receive ID for mailbox 16, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 17 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 17 | Select the receive ID for mailbox 17, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 18 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 18 | Select the receive ID for mailbox 18, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 19 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 19 | Select the receive ID for mailbox 19, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |

Input > Mailbox 16-19 Group > Mailbox ID Mode

| | | | |
|---|---|---|---|
| Mailbox 16 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 17 ID Mode | • Standard ID | Standard ID | Select whether the |

| | | | |
|---|---|---|---|
| | • Extended ID | | mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 18 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 19 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 16-19 Group > Mailbox Type | | | |
| Mailbox 16 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 17 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 18 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 19 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 16-19 Group > Mailbox Frame Type | | | |
| Mailbox 16 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 17 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to |

| | | | |
|---|---|---|---|
| | | | capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 18 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 19 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 16-19 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 16-19. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |
| Input > Mailbox 20-23 Group | | | |
| Input > Mailbox 20-23 Group > Mailbox ID | | | |
| Mailbox 20 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 20 | Select the receive ID for mailbox 20, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 21 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 21 | Select the receive ID for mailbox 21, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 22 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 22 | Select the receive ID for mailbox 22, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using |

| | | | |
|---|---|---|---|
| | | | extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 23 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 23 | Select the receive ID for mailbox 23, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 20-23 Group > Mailbox ID Mode | | | |
| Mailbox 20 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 21 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 22 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 23 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 20-23 Group > Mailbox Type | | | |
| Mailbox 20 Type | • Receive Mailbox | Receive Mailbox | Select whether the mailbox is used for |

| | | | |
|---|---|---|---|
| | • Transmit Mailbox | | receive or transmit. |
| Mailbox 21 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 22 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 23 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 20-23 Group > Mailbox Frame Type | | | |
| Mailbox 20 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 21 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 22 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 23 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 20-23 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 20-23. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |
| Input > Mailbox 24-27 Group | | | |
| Input > Mailbox 24-27 Group > Mailbox ID | | | |

| | | | |
|---|---|---|---|
| Mailbox 24 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 24 | Select the receive ID for mailbox 24, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 25 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 25 | Select the receive ID for mailbox 25, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 26 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 26 | Select the receive ID for mailbox 26, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 27 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 27 | Select the receive ID for mailbox 27, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 24-27 Group > Mailbox ID Mode | | | |
| Mailbox 24 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 25 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to |

| | | | |
|---|---|---|---|
| | | | receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 26 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 27 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 24-27 Group > Mailbox Type | | | |
| Mailbox 24 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 25 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 26 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 27 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 24-27 Group > Mailbox Frame Type | | | |
| Mailbox 24 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 25 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or |

| | | | |
|---|---|---|---|
| | | | remote frames (ignored for transmit mailboxes). |
| Mailbox 26 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 27 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 24-27 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 24-27. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |
| Input > Mailbox 28-31 Group | | | |
| Input > Mailbox 28-31 Group > Mailbox ID | | | |
| Mailbox 28 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 28 | Select the receive ID for mailbox 28, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 29 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 29 | Select the receive ID for mailbox 29, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Mailbox 30 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 30 | Select the receive ID for mailbox 30, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is |

| | | | |
|---|---|---|---|
| | | | not used when the mailbox is set as transmit type. |
| Mailbox 31 ID | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 31 | Select the receive ID for mailbox 31, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type. |
| Input > Mailbox 28-31 Group > Mailbox ID Mode | | | |
| Mailbox 28 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 29 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 30 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Mailbox 31 ID Mode | • Standard ID<br>• Extended ID | Standard ID | Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'. |
| Input > Mailbox 28-31 Group > Mailbox Type | | | |
| Mailbox 28 Type | • Receive Mailbox<br>• Transmit | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |

| | | | |
|---|---|---|---|
| | Mailbox | | |
| Mailbox 29 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 30 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Mailbox 31 Type | • Receive Mailbox<br>• Transmit Mailbox | Receive Mailbox | Select whether the mailbox is used for receive or transmit. |
| Input > Mailbox 28-31 Group > Mailbox Frame Type | | | |
| Mailbox 28 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 29 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 30 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 31 Frame Type | • Data Mailbox<br>• Remote Mailbox | Data Mailbox | Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes). |
| Mailbox 28-31 Group Mask | Value must be decimal or HEX integer of 0x1FFFFFFF or less. | 0x1FFFFFFF | Select the Mask for mailboxes 28-31. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value. |

### Clock Configuration

The CAN peripheral uses the CANMCLK (main-clock oscillator) or PCLKB as its clock source (fCAN, CAN System Clock.) The default CAN configuration will provide a CAN bit rate of 500 Kbit using

CANMCLK as the clock source. To set the PCLKB frequency, use the **Clocks** tab of the RA Configuration editor. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

Warning

RA2 devices only support CANMCLK (MOSC/XTAL) for the CAN clock source. If MOSC is not used as the main clock source it will not be started automatically. In this case, be sure to start it before opening the CAN driver.

### Clock Limitations

The following clock limitations apply when using the CAN peripheral:

- When using the main oscillator (CANMCLK) as the clock source:
  - fPCLKB >= fCANCLK (fCANCLK = XTAL / Baud Rate Prescaler)
  - The user application must start the main-clock oscillator (XTAL) at run-time using the CGC Interface if it has not already started (for example, if it is not used as the MCU clock source.)
- When using PCLKB as the clock source:
  - For RA6 and RA4 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module.
- For RA4M1 and RA4W1 MCUs, the clock frequency ratio of PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For RA2 MCUs only CANMCLK (XTAL) may be used as a clock source. The clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.

  *Note*

  *When using CANMCLK (XTAL) as the CAN clock source while running at a reduced main clock speed (under 2x XTAL) be sure to confirm that the XTAL frequency divided by the baud rate prescaler is equal to or less than PCLKB.*

### Pin Configuration

The CAN peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. A CAN channel would consist of two pins - CRX and CTX for data transmission/reception.

## Usage Notes

### Bit Rate Calculation

For convenience, the baudrate of the CAN peripheral is automatically set through the RA Configuration editor using a best effort approach. If the auto-generated baud settings cause deviation that is not tolerable by the application, the user can override the auto-generated settings and put in manually calculated values through RA Configuration editor. For more details on how the baudrate is set refer to section "Data Transfer Rate Configuration" in the CAN section of the relevant hardware manual.

### FIFO Support

When FIFO Support is enabled, mailboxes 24-27 form a 4-stage transmit FIFO and mailboxes 28-31 form a 4-stage receive FIFO. The receive FIFO supports two independent ID/mask settings for acceptance filtering.

*Note*

*Only the base mailbox of each FIFO may be accessed. When writing to the TX FIFO it is recommended to use CAN_MAILBOX_ID_TX_FIFO.*

### Limitations

Developers should be aware of the following limitations when using CAN:

- The can_frame_t::id_mode field is only used when Global ID Mode is set to Mixed ID. It is ignored in all other modes.

## Examples

### Basic Example

This is a basic example of minimal use of the CAN in an application.

```
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile bool     g_rx_flag = false;
volatile bool     g_tx_flag = false;
volatile bool     g_err_flag = false;
volatile uint32_t g_rx_id;
void can_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
     {
           g_rx_flag = true;
           g_rx_id   = p_args->frame.id;
 /* Read received frame */
           g_can_rx_frame = p_args->frame;
 break;
      }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
     {
           g_tx_flag = true;
 break;
      }
```

```
 case CAN_EVENT_ERR_BUS_OFF:          /* Bus error event. (bus off) */
 case CAN_EVENT_ERR_PASSIVE:          /* Bus error event. (error passive) */
 case CAN_EVENT_ERR_WARNING:          /* Bus error event. (error warning) */
 case CAN_EVENT_BUS_RECOVERY:         /* Bus error event. (bus recovery) */
 case CAN_EVENT_MAILBOX_MESSAGE_LOST: /* Overwrite/overrun error */
      {
 /* Set error flag */
            g_err_flag = true;
 break;
       }
 default:
      {
 break;
       }
    }
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = CAN_BUSY_DELAY;
 /* Initialize the CAN module */
    err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    g_can_tx_frame.id              = CAN_DESTINATION_DEVICE_MAILBOX_NUMBER; /* CAN
Destination Device ID */
    g_can_tx_frame.type            = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
```

```
/* Send data on the bus */
   g_tx_flag = false;
   g_err_flag = false;
   err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers*/
while ((true != g_tx_flag) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
if (true == g_err_flag)
   {
      __BKPT(0);
   }
}
```

### External Loop-back Test

This example requires the CTX and CRX pins to be connected. If a CAN tranciever is onboard a 120 Ohm resistor should be connected across CANH and CANL instead. The mailbox numbers are arbitrarily chosen.

```
void can_external_loopback_example (void)
{
fsp_err_t            err;
   uint32_t             timeout_ms    = CAN_BUSY_DELAY;
can_operation_mode_t operation_mode = CAN_OPERATION_MODE_NORMAL;
can_test_mode_t      test_mode      = CAN_TEST_MODE_LOOPBACK_EXTERNAL;
int      diff = 0;
   uint32_t i    = 0;
   err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
   err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
   assert(FSP_SUCCESS == err);
```

```
 /* Clear the data part of receive frame */
    memset(g_can_rx_frame.data, 0, CAN_FRAME_TRANSMIT_DATA_BYTES);
 /* CAN Destination Device ID, in this case it is the same device with another
mailbox */
    g_can_tx_frame.id              = CAN_MAILBOX_NUMBER_4;
    g_can_tx_frame.type            = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    g_rx_flag = false;
    g_err_flag = false;
    err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_rx_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (true == g_err_flag)
    {
       __BKPT(0);
    }
 /* Verify received data */
    diff = memcmp(&g_can_rx_frame.data[0], &g_can_tx_frame.data[0],
CAN_FRAME_TRANSMIT_DATA_BYTES);
 if (0 != diff)
    {
       __BKPT(0);
    }
```

```
}
```

### Data Structures

| | |
|---|---|
| struct | can_mailbox_t |
| struct | can_fifo_interrupt_cfg_t |
| struct | can_rx_fifo_cfg_t |
| struct | can_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | can_status_t |
| enum | can_error_t |
| enum | can_mailbox_number_t |
| enum | can_mailbox_send_receive_t |
| enum | can_global_id_mode_t |
| enum | can_message_mode_t |
| enum | can_clock_source_t |
| enum | can_fifo_interrupt_mode_t |

## Data Structure Documentation

### ◆ can_mailbox_t

| struct can_mailbox_t | | |
|---|---|---|
| CAN Mailbox | | |
| Data Fields | | |
| uint32_t | mailbox_id | Mailbox ID. |
| can_id_mode_t | id_mode | Standard or Extended ID. Only used in Mixed ID mode. |
| can_frame_type_t | frame_type | Frame type for receive mailbox. |
| can_mailbox_send_receive_t | mailbox_type | Receive or Transmit mailbox type. |

### ◆ can_fifo_interrupt_cfg_t

| struct can_fifo_interrupt_cfg_t |
|---|
| |

| CAN FIFO interrupt configuration | | |
|---|---|---|
| Data Fields | | |
| can_fifo_interrupt_mode_t | fifo_int_mode | FIFO interrupts mode (RX and TX combined). |
| IRQn_Type | tx_fifo_irq | TX FIFO IRQ. |
| IRQn_Type | rx_fifo_irq | RX FIFO IRQ. |

### ◆ can_rx_fifo_cfg_t

| struct can_rx_fifo_cfg_t | | |
|---|---|---|
| CAN RX FIFO configuration | | |
| Data Fields | | |
| uint32_t | rx_fifo_mask1 | RX FIFO acceptance filter mask 1. |
| uint32_t | rx_fifo_mask2 | RX FIFO acceptance filter mask 1. |
| can_mailbox_t | rx_fifo_id1 | RX FIFO acceptance filter ID 1. |
| can_mailbox_t | rx_fifo_id2 | RX FIFO acceptance filter ID 2. |

### ◆ can_extended_cfg_t

| struct can_extended_cfg_t | | |
|---|---|---|
| CAN extended configuration | | |
| Data Fields | | |
| can_clock_source_t | clock_source | Source of the CAN clock. |
| uint32_t * | p_mailbox_mask | Mailbox mask, one for every 4 mailboxes. |
| can_mailbox_t * | p_mailbox | Pointer to mailboxes. |
| can_global_id_mode_t | global_id_mode | Standard or Extended ID mode. |
| uint32_t | mailbox_count | Number of mailboxes. |
| can_message_mode_t | message_mode | Overwrite message or overrun. |
| can_fifo_interrupt_cfg_t const * | p_fifo_int_cfg | Pointer to FIFO interrupt configuration. |
| can_rx_fifo_cfg_t * | p_rx_fifo_cfg | Pointer to RX FIFO configuration. |

## Enumeration Type Documentation

### ◆ can_status_t

| enum can_status_t | |
|---|---|
| CAN Status | |
| Enumerator | |
| CAN_STATUS_NEW_DATA | New Data status flag. |
| CAN_STATUS_SENT_DATA | Sent Data status flag. |
| CAN_STATUS_RECEIVE_FIFO | Receive FIFO status flag. |
| CAN_STATUS_TRANSMIT_FIFO | Transmit FIFO status flag. |
| CAN_STATUS_NORMAL_MBOX_MESSAGE_LOST | Normal mailbox message lost status flag. |
| CAN_STATUS_FIFO_MBOX_MESSAGE_LOST | FIFO mailbox message lost status flag. |
| CAN_STATUS_TRANSMISSION_ABORT | Transmission abort status flag. |
| CAN_STATUS_ERROR | Error status flag. |
| CAN_STATUS_RESET_MODE | Reset mode status flag. |
| CAN_STATUS_HALT_MODE | Halt mode status flag. |
| CAN_STATUS_SLEEP_MODE | Sleep mode status flag. |
| CAN_STATUS_ERROR_PASSIVE | Error-passive status flag. |
| CAN_STATUS_BUS_OFF | Bus-off status flag. |

### ◆ can_error_t

| enum can_error_t | |
|---|---|
| CAN Error Code | |
| Enumerator | |
| CAN_ERROR_STUFF | Stuff Error. |
| CAN_ERROR_FORM | Form Error. |
| CAN_ERROR_ACK | ACK Error. |
| CAN_ERROR_CRC | CRC Error. |
| CAN_ERROR_BIT_RECESSIVE | Bit Error (recessive) Error. |
| CAN_ERROR_BIT_DOMINANT | Bit Error (dominant) Error. |
| CAN_ERROR_ACK_DELIMITER | ACK Delimiter Error. |

### ◆ can_mailbox_number_t

| enum can_mailbox_number_t |
|---|
| CAN Mailbox IDs (MB + FIFO) |

### ◆ can_mailbox_send_receive_t

| enum can_mailbox_send_receive_t | |
|---|---|
| CAN Mailbox type | |
| Enumerator | |
| CAN_MAILBOX_RECEIVE | Mailbox is for receiving. |
| CAN_MAILBOX_TRANSMIT | Mailbox is for sending. |

### ◆ can_global_id_mode_t

| enum can_global_id_mode_t | |
|---|---|
| Global CAN ID mode settings | |
| Enumerator | |
| CAN_GLOBAL_ID_MODE_STANDARD | Standard IDs of 11 bits used. |
| CAN_GLOBAL_ID_MODE_EXTENDED | Extended IDs of 29 bits used. |
| CAN_GLOBAL_ID_MODE_MIXED | Both Standard and Extended IDs used. |

### ◆ can_message_mode_t

| enum can_message_mode_t | |
|---|---|
| CAN Message Modes | |
| Enumerator | |
| CAN_MESSAGE_MODE_OVERWRITE | Receive data will be overwritten if not read before the next frame. |
| CAN_MESSAGE_MODE_OVERRUN | Receive data will be retained until it is read. |

### ◆ can_clock_source_t

| enum can_clock_source_t | |
|---|---|
| CAN Source Clock | |
| Enumerator | |
| CAN_CLOCK_SOURCE_PCLKB | PCLKB is the source of the CAN Clock. |
| CAN_CLOCK_SOURCE_CANMCLK | CANMCLK is the source of the CAN Clock. |

### ◆ can_fifo_interrupt_mode_t

| enum can_fifo_interrupt_mode_t |
|---|
| CAN FIFO Interrupt Modes |

## Function Documentation

### ◆ R_CAN_Open()

fsp_err_t R_CAN_Open ( can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg* )

Open and configure the CAN channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully |
| FSP_ERR_ALREADY_OPEN | Driver already open. |
| FSP_ERR_CAN_INIT_FAILED | Channel failed to initialize. |
| FSP_ERR_ASSERTION | Null pointer presented. |

### ◆ R_CAN_Close()

fsp_err_t R_CAN_Close ( can_ctrl_t *const *p_api_ctrl*)

Close the CAN channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented. |

### ◆ R_CAN_Write()

fsp_err_t R_CAN_Write ( can_ctrl_t *const *p_api_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame* )

Write data to the CAN channel. Write up to eight bytes to the channel mailbox.

Example:

```
err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_CAN_TRANSMIT_NOT_READY | Transmit in progress, cannot write data at this time. |
| FSP_ERR_CAN_TRANSMIT_FIFO_FULL | Transmit FIFO is full. |
| FSP_ERR_CAN_RECEIVE_MAILBOX | Mailbox is setup for receive and cannot send. |
| FSP_ERR_INVALID_ARGUMENT | Data length or frame type invalid. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_Read()

fsp_err_t R_CAN_Read ( can_ctrl_t *const *p_api_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame* )

Read data from a mailbox or FIFO.

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported. |

### ◆ R_CAN_ModeTransition()

fsp_err_t R_CAN_ModeTransition ( can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode* )

CAN Mode Transition is used to change CAN driver state.

Example:

```
err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_InfoGet()

fsp_err_t R_CAN_InfoGet ( can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info* )

Get CAN state and status information for the channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CAN_CallbackSet()

| fsp_err_t R_CAN_CallbackSet ( can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void *const *p_context*, can_callback_args_t *const *p_callback_memory* ) |
|---|
| Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.6.3 CAN FD (r_canfd)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_CANFD_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg) |
| fsp_err_t | R_CANFD_Close (can_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_CANFD_Write (can_ctrl_t *const p_api_ctrl, uint32_t buffer, can_frame_t *const p_frame) |
| fsp_err_t | R_CANFD_Read (can_ctrl_t *const p_api_ctrl, uint32_t buffer, can_frame_t *const p_frame) |
| fsp_err_t | R_CANFD_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode) |
| fsp_err_t | R_CANFD_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info) |
| fsp_err_t | R_CANFD_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the CANFD peripheral on RA MCUs. This module implements the CAN Interface.

# Overview

The CANFD module can be used to communicate over CAN networks, optionally using Flexible Data (CAN-FD) to accelerate the data phase. A variety of message filtering and buffer options are available. This document covers both the CANFD (r_canfd) and CANFD Lite (r_canfdlite) modules as they are based on the same hardware.

### Features

### Common Features

- Compatibility
  - Send and receive CAN 2.0 and CAN-FD frames on the same channel
  - Bitrate up to 1 Mbps with FD data phase speeds up to 5-8 Mbps
  - ISO 11898-1 Support
- Buffers
  - 32 global receive Message Buffers (RX MBs)
  - 2-8 global receive FIFOs (RX FIFOs)
  - 4-16 transmit Message Buffers (TX MBs) per channel
- Filtering
  - Each filter rule can be individually configured to accept messages based on:
    - ID
    - Standard or Extended ID (IDE bit)
    - Data or Remote Frame (RTR bit)
    - ID/IDE/RTR mask
    - Minimum DLC (data length) value
- Interrupts
  - Configurable Global RX FIFO Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Configurable Common FIFO RX Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Channel TX Interrupt
    - Interrupt on every transmitted message or when a Common FIFO is empty
  - Global Error
    - DLC Check
    - Message Lost
    - FD Payload Overflow
  - Channel Error
    - Bus Error
    - Error Warning
    - Error Passive
    - Bus-Off Entry
    - Bus-Off Recovery
    - Overload
    - Bus Lock
    - Arbitration Loss
    - Transmission Aborted

### Per-MCU Specifications

| | | | |
|---|---|---|---|

| | RA6M5 | RA6T2 | RA8M1 |
|---|---|---|---|
| Channels | 2 | 1 | 2 |
| Max nominal bitrate | 1 Mbps | 1 Mbps | 1 Mbps |
| Max data bitrate | 5 Mbps | 5 Mbps | 8 Mbps |
| Filter rules | 128 | 32 | 16/ch |
| TX message buffers | 16/ch | 4 | 4/ch |
| RX message buffers | 32 | 32 | 16/ch |
| RX FIFOs | 8 | 2 | 2/ch |
| RX Buffer RAM | 4864 bytes | 1216 bytes | 1216 bytes |
| Common FIFOs | 3/ch | 1 | 1/ch |

*Note*

*Each message buffer comprises 12 header bytes plus data length (8-64 bytes). The above buffer RAM values therefore correspond to the following capacities:*

| | *RA6M5* | *RA6T2* | *RA8M1* |
|---|---|---|---|
| *Max 64-byte storage* | *64 messages* | *16 messages* | *16 messages* |
| *Max 8-byte storage* | *243 messages* | *60 messages* | *60 messages* |

*Each Common FIFO buffer can support the following message capacities:*

| | *RA6M5* | *RA6T2* | *RA8M1* |
|---|---|---|---|
| *Maximum payload size* | *64 bytes* | *64 bytes* | *64 bytes* |
| *Maximum FIFO depth* | *128 messages* | *48 messages* | *48 messages* |

# Configuration

### Build Time Configurations for r_canfd

The following build time configurations are defined in fsp_cfg/r_canfd_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Global Error Interrupt | | | |
| Callback Channel | MCU Specific Options | | Specify which channel callback should be called to handle global errors. When starting the driver this channel must be opened first. |
| Priority | MCU Specific Options | | This interrupt is fired for each of the error sources selected below. |

| | | | |
|---|---|---|---|
| Sources | • DLC Check<br>• Message Lost<br>• FD Payload Overflow | 0U | Select which errors should trigger an interrupt. |
| Reception | | | |
| Reception > FIFOs | | | |
| Reception > FIFOs > FIFO 0 | | | |
| Enable | • Enabled<br>• Disabled | Enabled | Enable or disable RX FIFO 0. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 0. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 0. |
| Reception > FIFOs > FIFO 1 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 1. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 1. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |

| | | | |
|---|---|---|---|
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 1. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 1. |
| Reception > FIFOs > FIFO 2 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 2. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 2. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 2. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 2. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 2. |
| Reception > FIFOs > FIFO 3 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 3. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 3. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 3. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message |

| | | | |
|---|---|---|---|
| | | | payload size for RX FIFO 3. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 3. |
| Reception > FIFOs > FIFO 4 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 4. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 4. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 4. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 4. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 4. |
| Reception > FIFOs > FIFO 5 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 5. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 5. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 5. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX |

| | | | |
|---|---|---|---|
| | | | FIFO 5. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 5. |
| Reception > FIFOs > FIFO 6 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 6. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 6. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 6. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 6. |
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 6. |
| Reception > FIFOs > FIFO 7 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable RX FIFO 7. |
| Interrupt Mode | MCU Specific Options | | Set the interrupt mode for RX FIFO 7. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for RX FIFO 7. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for RX FIFO 7. |

| | | | |
|---|---|---|---|
| Depth | MCU Specific Options | | Select the number of stages for RX FIFO 7. |
| Interrupt Priority | MCU Specific Options | | This priority level will apply to all FIFO interrupts globally. |
| Reception > Message Buffers | | | |
| Number of Buffers | RX Message Buffer number must be an integer between 0 and 32. | 0 | Number of message buffers available for reception.<br><br>As there is no interrupt for message buffer reception it is recommended to use RX FIFOs instead. Set this value to 0 to disable RX Message Buffers. |
| Payload Size | • 8 bytes<br>• 12 bytes<br>• 16 bytes<br>• 20 bytes<br>• 24 bytes<br>• 32 bytes<br>• 48 bytes<br>• 64 bytes | 8 bytes | Payload size for all RX Message Buffers. |
| Reception > Acceptance Filtering | | | |
| Channel 0 Rule Count | The number of AFL rules must be a positive integer. | 32 | Number of acceptance filter list rules dedicated to Channel 0. |
| Channel 1 Rule Count | The number of AFL rules must be a positive integer. | 0 | Number of acceptance filter list rules dedicated to Channel 1. |
| Flexible Data (FD) | | | |
| Protocol Exceptions | • Enabled (ISO 11898-1)<br>• Disabled | Enabled (ISO 11898-1) | Select whether to enter the protocol exception handling state when a RES bit is sampled recessive as defined in ISO 11898-1. |
| Payload Overflow | • Reject<br>• Truncate | Reject | Configure whether received messages larger than the destination buffer should be truncated or rejected. |

Common FIFOs

Common FIFOs > FIFO 0

| | | | |
|---|---|---|---|
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 0. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 0. |
| RX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message receiption for Common FIFO 0. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 0. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 0. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 0. |
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 0. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 0 in units of the Interval Timer Clock. |

| | | | |
|---|---|---|---|
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 0. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 0. |
| Common FIFOs > FIFO 1 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 1. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 1. |
| RX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message receiption for Common FIFO 1. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 1. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 1. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 1. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 1. |

| | | | |
|---|---|---|---|
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 1. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 1 in units of the Interval Timer Clock. |
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 1. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 1. |
| Common FIFOs > FIFO 2 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 2. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 2. |
| RX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message receiption for Common FIFO 2. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 2. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 2. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 2. This setting is only |

| | | | |
|---|---|---|---|
| | | | applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 2. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 2. |
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 2. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 2 in units of the Interval Timer Clock. |
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 2. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 2. |
| Common FIFOs > FIFO 3 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 3. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 3. |
| RX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message receiption for Common FIFO 3. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 3. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 3. Threshold mode will |

| | | | |
|---|---|---|---|
| | | | only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 3. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 3. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 3. |
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 3. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 3 in units of the Interval Timer Clock. |
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 3. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 3. |
| Common FIFOs > FIFO 4 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 4. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 4. |
| RX Interrupt Enable | • Enabled | Disabled | Enable to allow |

| | | | |
|---|---|---|---|
| | • Disabled | | interrupts on message receiption for Common FIFO 4. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 4. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 4. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 4. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 4. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 4. |
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 4. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 4 in units of the Interval Timer Clock. |
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 4. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer |

| | | | |
|---|---|---|---|
| | | | source for Common FIFO 4. |
| Common FIFOs > FIFO 5 | | | |
| Enable | • Enabled<br>• Disabled | Disabled | Enable or disable Common FIFO 5. |
| Mode | • Receive<br>• Transmit | Receive | Select the operation mode for Common FIFO 5. |
| RX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message receiption for Common FIFO 5. |
| TX Interrupt Enable | • Enabled<br>• Disabled | Disabled | Enable to allow interrupts on message transmission for Common FIFO 5. |
| Interrupt Mode | • Threshold<br>• Every frame | Threshold | Set the interrupt mode for Common FIFO 5. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below or when all messages have been transmitted. |
| Interrupt Threshold | MCU Specific Options | | Set the interrupt threshold value for Common FIFO 5. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'. |
| Payload Size | MCU Specific Options | | Select the message payload size for Common FIFO 5. |
| Depth | MCU Specific Options | | Select the number of stages for Common FIFO 5. |
| TX Message Buffer | MCU Specific Options | | Select the TX message buffer that participates in TX scans related to Common FIFO 5. |
| Interval TX Delay | TX delay must be an integer between 0 and 255. | 0 | Selects the delay between successive transmission for Common FIFO 5 in units of the Interval |

| | | | |
|---|---|---|---|
| | | | Timer Clock. |
| Interval Timer | • Reference clock<br>• Bit time clock | Reference clock | Select the source of the interval timer clock for Common FIFO 5. The reference clock can be scaled by x10 below. |
| Interval Timer Reference Clock Resolution | • x1<br>• x10 | x1 | Selects the timer resolution if the reference clock is selected as the timer source for Common FIFO 5. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Transmission Priority | • Message ID<br>• Buffer Number | Buffer Number | Select how messages should be prioritized for transmission. In either case, lower numbers indicate higher priority. |
| DLC Check | • Disabled<br>• Enabled<br>• Enabled w/truncate | config.driver.canfd.dlc_check.disabled | When enabled received messages will be rejected if their DLC field is less than the value configured in the associated AFL rule. If 'Enabled w/truncate' is selected and a message passes the DLC check the DLC field is set to the value in the associated AFL rule and any excess data is discarded. |
| Interval Timer Prescaler | Timer prescaler must be an integer between 0 and 65535. | 0 | FIFO interval timer prescaler, required for Interval TX Delay. |

**Configurations for Connectivity > CAN FD (r_canfd)**

This module can be added to the Stacks tab via New Stack > Connectivity > CAN FD (r_canfd). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_canfd0 | Module name. |

| | | | |
|---|---|---|---|
| Channel | Channel should be 0 or 1 | 0 | Specify the CAN channel to use. |
| Bitrate | | | |
| Bitrate > Automatic | | | |
| Nominal Rate (bps) | Must be a valid integer with a maximum of 1MHz. | 500000 | Specify nominal bitrate in bits per second. |
| FD Data Rate (bps) | Must be a valid integer | 2000000 | Specify data bitrate in bits per second. This value is not used when sending messages without the FD and BRS bits enabled and should be left at the default setting if only Classical CAN will be used. |
| Sample Point (%) | Must be a valid integer between 60 and 99. | 75 | Specify desired sample point. |
| Bitrate > Manual | | | |
| Bitrate > Manual > Nominal | | | |
| Prescaler (divisor) | Value must be a non-negative integer between 1 and 1024. | 1 | Specify clock divisor for nominal bitrate. |
| Time Segment 1 (Tq) | Value must be a non-negative integer between 2 and 256. | 29 | Select the Time Segment 1 value. Check module usage notes for how to calculate this value. |
| Time Segment 2 (Tq) | Value must be a non-negative integer between 2 and 128. | 10 | Select the Time Segment 2 value. Check module usage notes for how to calculate this value. |
| Sync Jump Width (Tq) | Value must be a non-negative integer between 1 and 128. | 4 | Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value. |
| Bitrate > Manual > Data | | | |
| Prescaler (divisor) | Value must be a non-negative integer between 1 and 256. | 1 | Specify clock divisor for data bitrate. |
| Time Segment 1 (Tq) | Value must be a non-negative integer | 5 | Select the Time Segment 1 value. |

| | | | |
|---|---|---|---|
| | between 2 and 32. | | Check module usage notes for how to calculate this value. |
| Time Segment 2 (Tq) | Value must be a non-negative integer between 2 and 16. | 2 | Select the Time Segment 2 value. Check module usage notes for how to calculate this value. |
| Sync Jump Width (Tq) | Value must be a non-negative integer between 1 and 16. | 1 | Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value. |
| Use manual settings | • Yes<br>• No | No | Select whether or not to override automatic baudrate generation and instead use the values specified here. |
| Delay Compensation | • Enable<br>• Disable | Enable | When enabled the CANFD module will automatically compensate for any transceiver or bus delay between transmitted and received bits. When manually supplying bit timing values with delay compensation enabled be sure the data prescaler is 2 or smaller for correct operation. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | canfd0_callback | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs. |
| Channel Interrupt Priority Level | MCU Specific Options | | Channel Error/Transmit interrupt priority. |
| Transmit Interrupts | MCU Specific Options | | Select which TX Message Buffers should trigger an interrupt when transmission is complete. |

| | | | |
|---|---|---|---|
| Channel Error Interrupts | • Error Warning<br>• Error Passive<br>• Bus-Off Entry<br>• Bus-Off Recovery<br>• Overload | 0U | Select which channel error interrupt sources to enable. |
| Filter List Array | Name must be a valid C symbol | p_canfd0_afl | Acceptance Filter List (AFL) rule array symbol name. |

### Clock Configuration

The CANFD peripheral uses PLL or PLL2 as its clock source. The RA Configuration editor will attempt to get as close as possible to the supplied bitrate with the configured clock source. To achieve an exact bitrate the CANFD source clock or divisor may need to be adjusted to meet the criteria in the formula below:

```
bitrate = canfd_clock_hz / ((time_segment_1 + time_segment_2 + 1) * prescalar)
```

For CANFD, the possible values for each element are as follows:

| Element | Min | Max (Nominal) | Max (Data) |
|---|---|---|---|
| Bitrate | - | 1 Mbps | 5-8 Mbps* |
| Time Segment 1 | 2 Tq | 256 Tq | 32 Tq |
| Time Segment 2 | 2 Tq | 128 Tq | 16 Tq |
| Sync Jump Width | 1 Tq | Time Segment 2 | Time Segment 2 |
| Prescalar | 1 | 1024 | 256 |

- RA6 devices support up to 5 Mbps; RA8 devices support up to 8 Mbps.

Use the **Clocks** tab of the RA Configuration editor to configure the CANFD clock source/divisor as well as to set the frequency of PLL or PLL2. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

### Pin Configuration

CANFD channels each control two pins: CRX (receive) and CTX (transmit).

# Usage Notes

### Buffers

The CANFD driver provides four types of buffers: Transmit Message Buffers (TX MBs), Receive Message Buffers (RX MBs), Receive FIFOs (RX FIFOs), and Common FIFOs (RX/TX Common FIFOs).

### TX Message Buffers

TX MBs are used for transmission only. Refer to the hardware manual for your device for information on which TX MBs are available.

*Note*

*The CANFD peripheral continually scans TX MBs for new data. Depending on the provided clock it may be possible to write to multiple TX MBs before transmission begins. In this case, messages will be sent in the priority specified by the Transmission Priority option in the RA Configuration editor.*

### RX Message Buffers

RX MBs are for reception only and may only hold one message at a time. The number of available RX MBs varies per device.

On RA6M5, RX MBs are shared between channels and no interrupts are provided. Use R_CANFD_InfoGet and R_CANFD_Read to poll and read them, respectively.

### RX FIFOs

RX FIFOs provide interrupt-driven queue functionality for receiving messages. All FIFOs have the following capabilities:

- Up to 64 byte payloads
- Up to 128 messages (RA6M5) or 48 messages for all other MCUs.
- Interrupt events:
  - On every received frame OR when filled to a specified fraction of its capacity
  - When a message is overwritten (message received on full FIFO)

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFD_InfoGet and R_CANFD_Read, respectively.

*Note*

*On the RA6M5, FIFOs are shared across all channels.*

### RX Buffer Pool

The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. The RA Configuration editor will provide a warning when the limit is exceeded.

The number of bytes used by RX MBs and individual FIFOs can be calculated as follows:

```
Total RX MB bytes used = (number of RX MBs enabled) * (RX MB payload size + 12 header bytes)
RX FIFO bytes used = (number of FIFO stages) * (FIFO payload size + 12 header bytes)
```

### Common FIFOs

Common FIFOs provide interrupt-driven queue functionality for receiving or transmitting messages. Unlike RX FIFOs, Common FIFOs are individual to each channel. Refer to the hardware manual for your device for information on how many Common FIFOs are available. Common FIFOs support the following capabilities:

- Either TX or RX behavior can be configured.
  - Each Common FIFO can only be configured as one or the other. It cannot operate in both TX and RX modes at the same time.
- Up to 64 byte payloads
- Up to 128 messages (RA6M5) or 48 messages on all other MCUs
- Interrupt events:
  - On every message received or when filled to a specified fraction of its capacity
  - On every message transmitted or when all messages have been transmitted and the FIFO is empty

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFD_InfoGet and R_CANFD_Read, respectively.

Users should be aware of the total memory allocated in the internal CAN-FD RAM area for Common FIFOs. Allocating too many FIFOs or entries can lead to unexpected behavior.

**Message Filtering (Acceptance Filter List)**

To filter messages to the desired message buffer or FIFO the CANFD peripheral uses an Acceptance Filter List (AFL). Each entry in the AFL provides a rule to check a message against along with destination and other filtering information. When a message is received the CANFD peripheral internally checks against every configured AFL rule for the channel. If a match is found the message is transferred to the destination(s) specified in the rule. The default template with one entry is shown below:

```
static const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH0_RULE_NUM] =
{
    {
       .id =
      {
 /* Specify the ID, ID type and frame type to accept. */
            .id         = 0x00000000,
            .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_EXTENDED,
       },
       .mask =
      {
 /* These values mask which ID/mode bits to compare when filtering messages. */
            .mask_id         = 0x1FFFFFFF,
            .mask_frame_type = 1,
            .mask_id_mode    = 1,
       },
```

```
      .destination =
     {
 /* If DLC checking is enabled any messages shorter than the below setting will be
rejected. */
           .minimum_dlc = CANFD_MINIMUM_DLC_0,
 /* Optionally specify a Receive Message Buffer (RX MB) to store accepted frames. RX
MBs do not have an
  * interrupt or overwrite protection and must be checked with R_CANFD_InfoGet and
R_CANFD_Read. */
           .rx_buffer   = CANFD_RX_MB_NONE,
 /* Specify which FIFO(s) to send filtered messages to. Multiple FIFOs can be OR'd
together. */
           .fifo_select_flags = CANFD_RX_FIFO_0,
      }
   }
};
```

AFL templates can be easily added to a project using the Developer Assistance feature in e² studio. Once the CANFD module is added to a project, drag and drop the elements circled below to build a filter list:

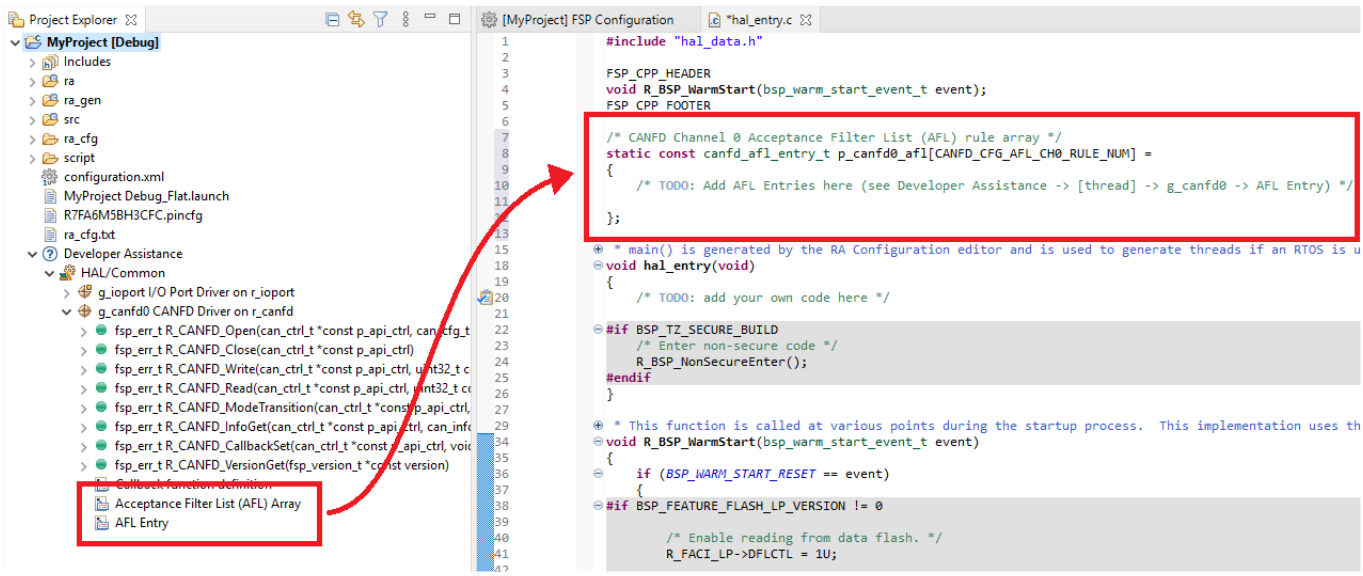



Figure 220: CANFD Developer Assistance AFL Templates

For an example configuration refer to the AFL Example below.

### Flexible Data (FD)

Flexible Data is an extension of the CAN protocol allowing for messages up to 64 bytes and higher data bitrates, among other features. The CANFD driver supports the following:

- Sending and receiving FD messages
- Bitrate switching for data phase
- Manual and automatic setting of the error state (ESI) bit

To specify one or more of these options when transmitting set can_frame_t::options with combined values from canfd_frame_options_t. Received messages will automatically have this field filled, if applicable.

```
/* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
   g_can_tx_frame.id               = CAN_EXAMPLE_ID;
   g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
   g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
   g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
   g_can_tx_frame.options          = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;
```

*Note*

*When using bitrate switching be sure to configure the Data Bitrate as desired in the RA Configuration editor.*

### Bit Rate Calculation

For convenience, the baudrate of the CANFD peripheral is automatically set through the RA Configuration editor using a best effort approach.

Enabling Delay Compensation instructs the CANFD peripheral to measure TX to RX tranceiver delay and automatically adjust for it, improving the reliability of high-speed FD messages. This option may severely limit available bitrate settings depending on the source clock; it is highly recommended to check the generated values when enabled.

If the auto-generated baud settings cause deviation that is not tolerable by the application the user can override the auto-generated settings and put in manually calculated values through the RA Configuration editor. For more details on how the bitrate is calculated refer to the Clock Configuration section above.

### Sync Jump Width

The Sync Jump Width option specifies the maximum number of time quanta that the sample point may be delayed by to account for differences in oscillators on the bus. It should be set to a value between 1 and the configured Time Segment 2 value depending on the maximum permissible clock error.

### Error Handling

The CANFD peripheral provides two types of error interrupts: Channel and Global. As the names imply, each channel has its own Channel Error interrupt but there is only one Global Error interrupt.

Only the configured channel will receive callbacks for Global Errors.

Error interrupt callbacks will pass either CAN_EVENT_ERR_CHANNEL or CAN_EVENT_ERR_GLOBAL in the can_callback_args_t::event field. A second field, can_callback_args_t::error, provides the actual error code as canfd_error_t. Cast to this enum to retrieve the error condition. See the callback in the Basic Example below for a demonstration.

### DLC Checking

When DLC Checking is enabled messages are checked against the destination.minimum_dlc value of each AFL rule. If the data length of a message is less than this value the message will be rejected. When DLC checking is set to "Enabled w/truncate" in the RA Configuration editor any data in excess of the minimum DLC setting will be truncated and the DLC value for the frame will be set to match.

### FD Payload Overflow

When an FD message is received with a DLC larger than the destination buffer an FD Payload Overflow interrupt is thrown (if configured). When Payload Overflow is set to "Truncate" the message will still be accepted but only data up to the buffer capacity will be preserved. The DLC value is unchanged in this case; any data beyond this value in the can_frame_t::data array should not be used.

### Test Modes

The CANFD peripheral provides three basic test modes: Listen Only, Internal Loopback and External Loopback. Use R_CANFD_ModeTransition to switch to a test mode.

On some MCUs an additional "Internal Bus" test mode is available that allows connecting both CANFD channels together on an internal bus, effectively creating an internal CAN network. See the Internal Bus example below for details.

### Limitations

Developers should be aware of the following limitations when using CANFD:

- On RA6M5, RX Message Buffers do not have an associated interrupt. To use them in an application one of the following is recommended:
  - Use R_CANFD_InfoGet to determine if any RX MBs have received data, then use R_CANFD_Read to obtain it
  - Select an RX FIFO as an additional destination for the relevant filter rules and configure the FIFO interrupt/callback as desired
- The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. See the RX Buffer Pool section above for more information.
- When switching modes with R_CANFD_ModeTransition a delay of up to several CAN frames may be incurred. Consult Section "Timing of Channel Mode Change" in the CANFD section of the relevant hardware manual for details.
- Only one channel will receive callbacks for Global Errors. If a different channel is opened first these error interrupts will be suppressed until the specified handler channel is opened.
- The CANFD driver does not currently support updating AFL entries in Operation mode (communication active). Please refer to the "Updating AFL entry during communication" section in your device's manual on how to use this feature (if available).

# Examples

### AFL Example

The below is an example Acceptance Filter List (AFL) declaration with two rules.

```
const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH1_RULE_NUM] =
{
 /* Store all data frames with at least 4 bytes from Standard IDs 0x40-0x4F in RX
FIFO 0 and RX FIFO 1 */
    {
       .id =
      {
            .id         = 0x40,
            .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_STANDARD
       },
       .mask =
      {
            .mask_id         = 0x7F0,
            .mask_frame_type = 1,
            .mask_id_mode    = 1
       },
       .destination =
      {
            .minimum_dlc       = CANFD_MINIMUM_DLC_4,
            .rx_buffer         = CANFD_RX_MB_NONE,
            .fifo_select_flags = (canfd_rx_fifo_t) (CANFD_RX_FIFO_0 |
CANFD_RX_FIFO_1)
       }
    },
 /* Store all frames from Extended ID 0x1100 in RX FIFO 2 and RX MB 0 */
    {
       .id =
      {
            .id         = 0x1100,
            .frame_type = CAN_FRAME_TYPE_DATA, // This setting is ignored by the mask
            .id_mode    = CAN_ID_MODE_EXTENDED
```

```
      },
      .mask =
     {
           .mask_id         = 0x1FFFFFFF,
           .mask_frame_type = 0,
           .mask_id_mode    = 1
      },
      .destination =
     {
           .minimum_dlc       = CANFD_MINIMUM_DLC_0,
           .rx_buffer         = CANFD_RX_MB_0,
           .fifo_select_flags = CANFD_RX_FIFO_2
      }
   }
};
```

### Basic Example

This is a basic example of minimal use of the CANFD module in an application.

*Note*

*On RA6M5 it is recommended to use RX FIFOs for reception as there are no interrupts for RX message buffers.*

```
#define CAN_EXAMPLE_ID (0x20)
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile canfd_error_t g_err_status = (canfd_error_t) 0;
void canfd_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
     {
 /* Read received frame */
           memcpy(&g_can_rx_frame, &p_args->frame, sizeof(can_frame_t));
 /* Handle event */
```

```
 break;
       }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
      {
 /* Handle event */
 break;
       }
 case CAN_EVENT_ERR_GLOBAL:                        /* Global error. */
 case CAN_EVENT_ERR_CHANNEL:                      /* Channel error. */
      {
 /* Get error status */
            g_err_status = (canfd_error_t) p_args->error; /* Check error code with
canfd_error_t. */
 /* Handle event */
 break;
       }
 default:
      {
 break;
       }
    }
}
void canfd_basic_example (void)
{
 fsp_err_t err;
 /* Initialize the CAN module */
    err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Setup frame to write to CAN ID 0x20 */
    g_can_tx_frame.id              = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode         = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type            = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = 8;
```

```
    g_can_tx_frame.options          = 0;
 /* Write some data to the transmit frame */
 for (uint32_t i = 0; i < 8; i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Wait for a transmit callback event */
}
```

### Flexible Data

This example demonstrates sending an FD message with bitrate switching over external loopback. The CTX and CRX pins must be connected when using external loopback, though if a CAN tranciever is onboard a 120 Ohm resistor should be connected across CANH and CANL instead.

```
#define CAN_EXAMPLE_64_BYTES 64
void canfd_fd_loopback_example (void)
{
 fsp_err_t err;
    err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Switch to external loopback mode */
    err = R_CANFD_ModeTransition(&g_canfd0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
    assert(FSP_SUCCESS == err);
 /* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
    g_can_tx_frame.options          = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;
 /* Write some data to the transmit frame */
```

```
for (uint32_t i = 0; i < CAN_DATA_BUFFER_LENGTH; i++)
   {
      g_can_tx_frame.data[i] = (uint8_t) i;
   }
/* Send data on the bus */
   err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
/* Wait for a transmit and/or receive callback event */
}
```

### Internal Bus

In this example two CANFD channels are connected to the Internal Bus test mode. API error checking has been omitted for clarity.

*Note*

*Internal Bus mode is only available on MCUs with more than one CANFD channel. In addition, use of Global Modes for any other purpose is not recommended without consulting the device User's Manual.*

### Data Structures

| | |
|---|---|
| struct | canfd_afl_entry_t |
| struct | canfd_global_cfg_t |
| struct | canfd_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | canfd_status_t |
| enum | canfd_error_t |
| enum | canfd_tx_buffer_t |
| enum | canfd_tx_mb_t |
| enum | canfd_rx_buffer_t |
| enum | canfd_rx_mb_t |
| enum | canfd_rx_fifo_t |
| enum | canfd_minimum_dlc_t |

| | |
|---|---|
| enum | canfd_frame_options_t |

## Data Structure Documentation

### ◆ canfd_afl_entry_t

| struct canfd_afl_entry_t |
|---|
| AFL Entry (based on R_CANFD_CFDGAFL_Type in renesas.h) |

### ◆ canfd_global_cfg_t

| struct canfd_global_cfg_t | | |
|---|---|---|
| CANFD Global Configuration | | |
| Data Fields | | |
| uint32_t | global_interrupts | Global control options (CFDGCTR register setting) |
| uint32_t | global_config | Global configuration options (CFDGCFG register setting) |
| uint32_t | rx_fifo_config[8] | RX FIFO configuration (CFDRFCCn register settings) |
| uint32_t | rx_mb_config | Number and size of RX Message Buffers (CFDRMNB register setting) |
| uint8_t | global_err_ipl | Global Error interrupt priority. |
| uint8_t | rx_fifo_ipl | RX FIFO interrupt priority. |
| uint32_t | common_fifo_config[R_CANFD_NUM_COMMON_FIFOS] | Common FIFO configurations. |

### ◆ canfd_extended_cfg_t

| struct canfd_extended_cfg_t | | |
|---|---|---|
| CANFD Extended Configuration | | |
| Data Fields | | |
| canfd_afl_entry_t const * | p_afl | AFL rules list. |
| uint64_t | txmb_txi_enable | Array of TX Message Buffer enable bits. |
| uint32_t | error_interrupts | Error interrupt enable bits. |
| can_bit_timing_cfg_t * | p_data_timing | FD Data Rate (when bitrate switching is used) |
| uint8_t | delay_compensation | FD Transceiver Delay Compensation (enable or disable) |
| canfd_global_cfg_t * | p_global_cfg | Global configuration (global |

| | | |
|---|---|---|
| | | error callback channel only) |

## Enumeration Type Documentation

### ◆ canfd_status_t

| enum canfd_status_t | |
|---|---|
| CANFD Status | |
| Enumerator | |
| CANFD_STATUS_RESET_MODE | Channel in Reset mode. |
| CANFD_STATUS_HALT_MODE | Channel in Halt mode. |
| CANFD_STATUS_SLEEP_MODE | Channel in Sleep mode. |
| CANFD_STATUS_ERROR_PASSIVE | Channel in error-passive state. |
| CANFD_STATUS_BUS_OFF | Channel in bus-off state. |
| CANFD_STATUS_TRANSMITTING | Channel is transmitting. |
| CANFD_STATUS_RECEIVING | Channel is receiving. |
| CANFD_STATUS_READY | Channel is ready for communication. |
| CANFD_STATUS_ESI | At least one CAN-FD message was received with the ESI flag set. |

### ◆ canfd_error_t

| enum canfd_error_t | |
|---|---|
| CANFD Error Code | |
| Enumerator | |
| CANFD_ERROR_CHANNEL_BUS | Bus Error. |
| CANFD_ERROR_CHANNEL_WARNING | Error Warning (TX/RX error count over 0x5F) |
| CANFD_ERROR_CHANNEL_PASSIVE | Error Passive (TX/RX error count over 0x7F) |
| CANFD_ERROR_CHANNEL_BUS_OFF_ENTRY | Bus-Off State Entry. |
| CANFD_ERROR_CHANNEL_BUS_OFF_RECOVERY | Recovery from Bus-Off State. |
| CANFD_ERROR_CHANNEL_OVERLOAD | Overload. |

| | |
|---|---|
| CANFD_ERROR_CHANNEL_BUS_LOCK | Bus Locked. |
| CANFD_ERROR_CHANNEL_ARBITRATION_LOSS | Arbitration Lost. |
| CANFD_ERROR_CHANNEL_STUFF | Stuff Error. |
| CANFD_ERROR_CHANNEL_FORM | Form Error. |
| CANFD_ERROR_CHANNEL_ACK | ACK Error. |
| CANFD_ERROR_CHANNEL_CRC | CRC Error. |
| CANFD_ERROR_CHANNEL_BIT_RECESSIVE | Bit Error (recessive) Error. |
| CANFD_ERROR_CHANNEL_BIT_DOMINANT | Bit Error (dominant) Error. |
| CANFD_ERROR_CHANNEL_ACK_DELIMITER | ACK Delimiter Error. |
| CANFD_ERROR_GLOBAL_DLC | DLC Error. |
| CANFD_ERROR_GLOBAL_MESSAGE_LOST | Message Lost. |
| CANFD_ERROR_GLOBAL_PAYLOAD_OVERFLOW | FD Payload Overflow. |
| CANFD_ERROR_GLOBAL_TXQ_OVERWRITE | TX Queue Message Overwrite. |
| CANFD_ERROR_GLOBAL_TXQ_MESSAGE_LOST | TX Queue Message Lost. |
| CANFD_ERROR_GLOBAL_CH0_SCAN_FAIL | Channel 0 RX Scan Failure. |
| CANFD_ERROR_GLOBAL_CH1_SCAN_FAIL | Channel 1 RX Scan Failure. |
| CANFD_ERROR_GLOBAL_CH0_ECC | Channel 0 ECC Error. |
| CANFD_ERROR_GLOBAL_CH1_ECC | Channel 1 ECC Error. |

### ◆ canfd_tx_buffer_t

| enum canfd_tx_buffer_t |
|---|
| CANFD Transmit Buffer (MB + CFIFO) |

### ◆ canfd_tx_mb_t

| enum canfd_tx_mb_t |
| --- |
| CANFD Transmit Message Buffer (TX MB) |

### ◆ canfd_rx_buffer_t

| enum canfd_rx_buffer_t |
| --- |
| CANFD Receive Buffer (MB + FIFO + CFIFO) |

### ◆ canfd_rx_mb_t

| enum canfd_rx_mb_t |
| --- |
| CANFD Receive Message Buffer (RX MB) |

### ◆ canfd_rx_fifo_t

| enum canfd_rx_fifo_t |
| --- |
| CANFD Receive FIFO (RX FIFO) |

### ◆ canfd_minimum_dlc_t

| enum canfd_minimum_dlc_t |
| --- |
| CANFD AFL Minimum DLC settings |

### ◆ canfd_frame_options_t

| enum canfd_frame_options_t | |
| --- | --- |
| CANFD Frame Options | |
| Enumerator | |
| CANFD_FRAME_OPTION_ERROR | Error state set (ESI). |
| CANFD_FRAME_OPTION_BRS | Bit Rate Switching (BRS) enabled. |
| CANFD_FRAME_OPTION_FD | Flexible Data frame (FDF). |

## Function Documentation

### ◆ R_CANFD_Open()

fsp_err_t R_CANFD_Open ( can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg* )

Open and configure the CANFD channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ALREADY_OPEN | Driver already open. |
| FSP_ERR_IN_USE | Channel is already in use. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel does not exist on this MCU. |
| FSP_ERR_ASSERTION | A required pointer was NULL. |
| FSP_ERR_CAN_INIT_FAILED | The provided nominal or data bitrate is invalid. |
| FSP_ERR_CLOCK_INACTIVE | CANFD source clock is disabled (PLL or PLL2). |

### ◆ R_CANFD_Close()

fsp_err_t R_CANFD_Close ( can_ctrl_t *const *p_api_ctrl*)

Close the CANFD channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented. |

### ◆ R_CANFD_Write()

fsp_err_t R_CANFD_Write ( can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame* )

Write data to the CANFD channel.

Example:

```
/* Send data on the bus */
   err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_CAN_TRANSMIT_NOT_READY | Transmit in progress, cannot write data at this time. |
| FSP_ERR_INVALID_ARGUMENT | Data length or buffer number invalid. |
| FSP_ERR_INVALID_MODE | An FD option was set on a non-FD frame. |
| FSP_ERR_ASSERTION | One or more pointer arguments is NULL. |
| FSP_ERR_UNSUPPORTED | FD is not supported on this MCU. |

### ◆ R_CANFD_Read()

fsp_err_t R_CANFD_Read ( can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame* )

Read data from a CANFD Message Buffer or FIFO.

Example: snippet r_canfd_example.c R_CANFD_Read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | Buffer number invalid. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_frame is NULL. |
| FSP_ERR_BUFFER_EMPTY | Buffer or FIFO is empty. |

### ◆ R_CANFD_ModeTransition()

fsp_err_t R_CANFD_ModeTransition ( can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode* )

Switch to a different channel, global or test mode.

Example:

```
/* Switch to external loopback mode */
   err = R_CANFD_ModeTransition(&g_canfd0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |
| FSP_ERR_INVALID_MODE | Cannot change to the requested mode from the current global mode. |

### ◆ R_CANFD_InfoGet()

fsp_err_t R_CANFD_InfoGet ( can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info* )

Get CANFD state and status information for the channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | Null pointer presented |

### ◆ R_CANFD_CallbackSet()

| fsp_err_t R_CANFD_CallbackSet ( can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void *const *p_context*, can_callback_args_t *const *p_callback_memory* ) |
|---|
| Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.6.4 CEC (r_cec)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_CEC_Open (cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg) |
| fsp_err_t | R_CEC_MediaInit (cec_ctrl_t *const p_ctrl, cec_addr_t local_address) |
| fsp_err_t | R_CEC_Close (cec_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CEC_Write (cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size) |
| fsp_err_t | R_CEC_StatusGet (cec_ctrl_t *const p_ctrl, cec_status_t *const p_status) |
| fsp_err_t | R_CEC_CallbackSet (cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void *const p_context, cec_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the CEC peripheral on RA MCUs. This module implements the CEC Interface.

# Overview

The HDMI CEC HAL module provides a high-level API for CEC applications and supports the CEC peripheral available on RA microcontroller hardware. A user-callback function must be defined that the driver will invoke when data received, transmission complete, or error interrupts are received. The callback is passed a parameter which indicates the event as well as received data (if available).

### Features

- Conforms to High Definition Multimedia Interface (HDMI) Consumer Electronics Control (CEC) standard Ver. 1.4b.
- Full range of local address settings (TV, Recording Device, Playback Device, etc.)
- Data filtering based on matching destination address and local address.
- Supports a user-callback function (required), invoked when transmit, receive, or error interrupts are received.

# Configuration

### Build Time Configurations for r_cec

The following build time configurations are defined in fsp_cfg/r_cec_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| CEC Message Max Data Size | CEC message max data size must be a positive integer | 14 | Maximum Data Size for CEC Message Transmission/Reception. |

### Configurations for Connectivity > CEC (r_cec)

This module can be added to the Stacks tab via New Stack > Connectivity > CEC (r_cec). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_cec0 | Module name |
| Control Configuration | | | |
| Clock Select | • PCLKB / 32<br>• PCLKB / 64<br>• PCLKB / 128<br>• PCLKB / 256<br>• PCLKB / 512<br>• PCLKB / 1024<br>• CECCLK<br>• CECCLK / 256 | PCLKB / 1024 | CEC Clock Select Configuration |
| Ack Bit Timing Error | • Disabled | Enabled | CEC Ack Bit Timing |

| | | | |
|---|---|---|---|
| Enable | • Enabled | | Error Enable |
| Signal-Free Time Bit Width | • 3-data bit width<br>• 5-data bit width<br>• 7-data bit width<br>• Does not detect signal-free time. | 7-data bit width | Signal-Free Time Data Bit Width Select |
| Start Bit Error Detection Enable | • Disabled<br>• Enabled | Enabled | Enable to detect timing errors during start bit reception. |
| Bus Lock Detection Enable | • Disabled<br>• Enabled | Enabled | Enable to detect sticking of receive data to high or low. |
| Digital Filter Enable | • Disabled<br>• Enabled | Enabled | Enable to use a digital filter. |
| Long Bit Width Error Pulse Output Enable | • Disabled<br>• Enabled | Disabled | Enable to output an error handling pulse when a long bit width error is detected. |
| Start Detection Reception Restart Enable | • Disabled<br>• Enabled | Enabled | Enable to restart reception after a start bit error is detected. |

Bit Width Timing

Bit Width Timing > Transmit

| | | | |
|---|---|---|---|
| Start Bit Low Time | CEC transmission start bit low width setting must be a positive integer. | 180 | CEC transmission start bit low width setting (CEC Clock Cycles). |
| Start Bit Width Time | CEC transmission start bit high width setting must be a positive integer. | 220 | CEC transmission start bit high width setting (CEC Clock Cycles). |
| Logical Zero Low Time | CEC transmission logical zero low width setting must be a positive integer. | 73 | CEC transmission logical zero low width setting (CEC Clock Cycles). |
| Logical One Low Time | CEC transmission logical one low width setting must be a positive integer. | 29 | CEC transmission logical one low width setting (CEC Clock Cycles). |
| Overall Bit Width Time | CEC transmission overall data bit width time setting must be a positive integer. | 117 | CEC transmission overall data bit width time setting (CEC Clock Cycles). |

Bit Width Timing > Receive

| | | | |
|---|---|---|---|
| Data Sample Time | CEC reception data sampling time must be a positive integer. | 49 | CEC reception data sampling time setting (CEC Clock Cycles). |
| Data Bit Reference Width | CEC reception data bit reference width must be a positive integer. | 117 | CEC data bit reference width setting (CEC Clock Cycles). |
| Start Bit Low Min Time | CEC reception start bit minimum low width setting must be a positive integer. | 171 | CEC reception start bit minimum low width setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled. |
| Start Bit Low Max Time | CEC reception start bit maximum low width setting must be a positive integer. | 190 | CEC reception start bit maximum low width setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled. |
| Start Bit Min Time | CEC start bit minimum time setting must be a positive integer. | 210 | CEC start bit minimum time setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled. |
| Start Bit Max Time | CEC reception start bit maximum time setting must be a positive integer. | 229 | CEC start bit maximum time setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled. |
| Logical Zero Low Min Time | CEC reception logical zero minimum low width setting must be a positive integer. | 64 | CEC reception logical zero minimum low width setting (CEC Clock Cycles). |
| Logical Zero Low Max Time | CEC reception locical zero maximum low width setting must be a positive integer. | 83 | CEC reception logical zero maximum low width setting (CEC Clock Cycles). |
| Logical One Low Min Time | CEC reception logical one minimum low width setting must be a positive integer. | 20 | CEC reception logical one minimum low width setting (CEC Clock Cycles). |
| Logical One Low Max | CEC reception logical | 39 | CEC reception logical |

| | | | |
|---|---|---|---|
| Time | one maximum low width setting must be a positive integer. | | one maximum low width (CEC Clock Cycles). |
| Overall Bit Width Min Time | CEC reception overall minimum bit width setting must be a positive integer. | 100 | CEC reception overall minimum bit width setting (CEC Clock Cycles). |
| Overall Bit Width Max Time | CEC reception overall maximum bit width setting must be a positive integer. | 134 | CEC reception overall maximum bit width setting (CEC Clock Cycles). |
| Interrupts | | | |
| Interrupt Priority Level | MCU Specific Options | | Error/Data/Message interrupt priority level. |
| Callback Function | Callback Function must be a valid C symbol | g_rm_cec0_callback | Callback function |
| Communication Complete Interrupt Timing | • After Last Frame and Signal Free Time<br>• After Last Frame<br>• After Signal Free Time | After Last Frame and Signal Free Time | Communication Complete Interrupt (INTCE) Generation Timing Select |
| Address Mismatch Interrupt Enable | • Disabled<br>• Enabled | Disabled | Enable to generate an interrupt when the addresses do not match. |
| Data Interrupt Timing Selection | • EOM timing (9th bit of data)<br>• ACK Timing (10th bit of data) | EOM timing (9th bit of data) | INTDA reception interrupt timing selection (EOM or ACK). |

### Clock Configuration

The CEC peripheral uses the CECCLK or PCLKB as its clock source. To set the PCLKB frequency, use the Clocks tab of the RA Configuration editor.

*Note*

*The selected clock and configured divider must be configured in the range of 23.4375 to 78.125 kHz.*

### Pin Configuration

A CEC channel uses one data pin - CECIO for data transmission and reception.

The output type for each pin should be set to **n-ch open drain** for most hardware designs. This can be configured in **Pins** tab of the RA Configuration editor by selecting the pin under Pin Selection->Ports.

# Usage Notes

### CEC Device Addresses

The CEC standard provides 13 device addresses that may be requested based on a device's primary function. Use R_CEC_MediaInit to request a specific address before starting communication with other devices.

*Note*

*Address 0 is always the primary display (TV). Do not attempt to allocate this address unless your device is intended to function as a display.*

### Limitations

Developers should be aware of the following limitations when using the CEC module:

- R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after R_CEC_Open while the hardware initializes.
- The CECIO pin must be set to n-ch open drain mode.

# Examples

### Basic Example

This is a basic example of minimal use of the CEC in an application.

```
/***********************************************************************************************************************
 * Application defined callback
 * - May be assigned at compile-time via the e2 studio configuration tool or set at run-time via R_CEC_CallbackSet()
 **********************************************************************************************************************/
void cec_callback (cec_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CEC_EVENT_READY:
     {
 /* Application processing for address allocation success. */
 break;
      }
 case CEC_EVENT_TX_COMPLETE:
```

```
     {
 /* Any required processing after transmission has completed. */
 break;
      }
 case CEC_EVENT_ERR:
     {
 /* Error processing. See cec_error_t for possible errors. */
 break;
      }
 case CEC_EVENT_RX_DATA:
     {
 /* Application to store and process received data bytes. */
 break;
      }
 case CEC_EVENT_RX_COMPLETE:
     {
 /* Application processing for message reception complete. */
      }
   }
}
/***********************************************************************************
*********************************
 * Basic example
 ***********************************************************************************
*********************************/
#define CEC_TIMEOUT_MS (50)
#define CEC_MSG_STANDBY (0X36)  /* See CEC Specification for message definitions */
void basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the CEC module */
    err = R_CEC_Open(&g_cec0_ctrl, &g_cec0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize the CEC module and allocate an address */
```

```
    uint32_t timeout_ms = CEC_TIMEOUT_MS;
 do
    {
 /* R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after calling
R_CEC_Open */
       err = R_CEC_MediaInit(&g_cec0_ctrl, CEC_ADDR_TV);
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    } while ((FSP_ERR_IN_USE == err) && --timeout_ms);
    assert(timeout_ms);
    assert(FSP_SUCCESS == err);
 /* Wait for local address allocation and CEC bus to be free */
    cec_status_t status;
    err = R_CEC_StatusGet(&g_cec0_ctrl, &status);
 while ((FSP_SUCCESS == err) && (CEC_STATE_READY != status.state))
    {
      err = R_CEC_StatusGet(&g_cec0_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
 cec_message_t cec_msg;
    uint8_t       total_transmit_size;
    cec_msg.destination = CEC_ADDR_BROADCAST;       /* For this example, send message
to all devices on the bus */
    cec_msg.opcode      = CEC_MSG_STANDBY;          /* Send Standby Request */
    memset(cec_msg.data, 0U, sizeof(cec_msg.data)); /* See CEC Specification for
other message data structures */
    total_transmit_size = 2U;                      /* Total message size, including
header, opcode, and data */
 /* Send asynchronous message.
  * - Application will then be free for other processing while message is being sent.
  * - Do not modify the message buffer until transmission has completed. */
    err = R_CEC_Write(&g_cec0_ctrl, &cec_msg, total_transmit_size);
    assert(FSP_SUCCESS == err);
}
```

## Function Documentation

### ◆ R_CEC_Open()

fsp_err_t R_CEC_Open ( cec_ctrl_t *const *p_ctrl*, cec_cfg_t const *const *p_cfg* )

Open and configure the CEC module for operation.

Example:

```
/* Open the CEC module */
   err = R_CEC_Open(&g_cec0_ctrl, &g_cec0_cfg);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CEC Module opened successfully. |
| FSP_ERR_ALREADY_OPEN | Driver already open. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_IRQ_BSP_DISABLED | Interrupts are not enabled. |

### ◆ R_CEC_MediaInit()

fsp_err_t R_CEC_MediaInit ( cec_ctrl_t *const *p_ctrl*, cec_addr_t *local_address* )

Allocate provided CEC Local Address and Initialize the CEC module for operation.

***Note***

*After calling R_CEC_Open this function may return FSP_ERR_IN_USE for up to 45 milliseconds.*

Example:

```
/* Initialize the CEC module and allocate an address */
   uint32_t timeout_ms = CEC_TIMEOUT_MS;
do
   {
/* R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after calling
R_CEC_Open */
      err = R_CEC_MediaInit(&g_cec0_ctrl, CEC_ADDR_TV);
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
   } while ((FSP_ERR_IN_USE == err) && --timeout_ms);
   assert(timeout_ms);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CEC Module Initialized successfully. |
| FSP_ERR_ASSERTION | An input argument is invalid or callback has not been set. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_IN_USE | HDMI CEC Bus is currently in use. Try again later. |

### ◆ R_CEC_Close()

fsp_err_t R_CEC_Close ( cec_ctrl_t *const *p_ctrl*)

Close the CEC module.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CEC Module closed successfully. |
| FSP_ERR_ASSERTION | An input argument is invalid. |

### ◆ R_CEC_Write()

fsp_err_t R_CEC_Write ( cec_ctrl_t *const *p_ctrl*, cec_message_t const *const *p_message*, uint32_t *message_size* )

Write data to the CEC bus. Data transmission is asynchronous. Provided message buffer should not be modified until transmission is complete.

Data Transmission follows the pattern defined be the HDMI CEC Specification:

| Data | Description | Size |
|---|---|---|
| Start Bit | Managed by Hardware, per config | N/A |
| Header Block | Source/Destination Identifier | 1 Byte |
| Data Block 1 | Opcode Value (Optional) | 1 Byte |
| Data Block 2 | Operands (Optional) | Variable (0-14 Bytes Typical) |

Example:

```
cec_message_t cec_msg;
    uint8_t       total_transmit_size;
    cec_msg.destination = CEC_ADDR_BROADCAST;       /* For this example, send message
to all devices on the bus */
    cec_msg.opcode      = CEC_MSG_STANDBY;          /* Send Standby Request */
    memset(cec_msg.data, 0U, sizeof(cec_msg.data)); /* See CEC Specification for
other message data structures */
    total_transmit_size = 2U;                      /* Total message size, including
header, opcode, and data */
 /* Send asynchronous message.
  * - Application will then be free for other processing while message is being sent.
  * - Do not modify the message buffer until transmission has completed. */
    err = R_CEC_Write(&g_cec0_ctrl, &cec_msg, total_transmit_size);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been successfully initialized. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_INVALID_SIZE | Invalid message size. |
| FSP_ERR_IN_USE | HDMI CEC Bus is currently in use. Try again |

| | |
|---|---|
| | later. |

### ◆ R_CEC_StatusGet()

fsp_err_t R_CEC_StatusGet ( cec_ctrl_t *const *p_ctrl*, cec_status_t *const *p_status* )

Provides the state and status information according to the provided CEC control instance.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_ASSERTION | An input argument is invalid. |

### ◆ R_CEC_CallbackSet()

fsp_err_t R_CEC_CallbackSet ( cec_ctrl_t *const *p_ctrl*, void(*)(cec_callback_args_t *) *p_callback*, void *const *p_context*, cec_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements cec_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | An input argument is invalid. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.6.5 I2C Communication Device (rm_comms_i2c)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | RM_COMMS_I2C_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg)<br><br>Opens and configures the Communications Middle module. Implements rm_comms_api_t::open. More... |

| | |
|---|---|
| fsp_err_t | RM_COMMS_I2C_Close (rm_comms_ctrl_t *const p_api_ctrl)<br>Disables specified Communications Middle module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_I2C_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)<br>Performs a read from the I2C device. Implements rm_comms_api_t::read. More... |
| fsp_err_t | RM_COMMS_I2C_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write from the I2C device. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_I2C_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs a write to, then a read from the I2C device. Implements rm_comms_api_t::writeRead. More... |
| void | rm_comms_i2c_callback (i2c_master_callback_args_t *p_args)<br>Common callback function called in the I2C driver callback function. |

### Detailed Description

Middleware to implement the I2C communications interface. This module implements the Communications Middleware Interface.

# Overview

### Features

The implementation of the I2C communications interface has the following key features:

- Reading data from, writing data to I2C bus
- Writes to I2C bus, then reads with restart
- A single I2C bus used by multiple I2C devices

# Configuration

### Build Time Configurations for rm_comms_i2c

The following build time configurations are defined in fsp_cfg/rm_comms_i2c_cfg.h:

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Connectivity > I2C Shared Bus (rm_comms_i2c)**

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Shared Bus (rm_comms_i2c).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comms_i2c_bus0 | Module name. |
| Bus Timeout | Value must be a non-negative integer less than or equal to 0xFFFFFFFF | 0xFFFFFFFF | Set timeout for locking bus in using RTOS. |
| Semaphore for Blocking (RTOS only) | • Unuse<br>• Use | Use | Set Semaphore for blocking in using RTOS. |
| Recursive Mutex for Bus (RTOS only) | • Unuse<br>• Use | Use | Set Mutex for locking bus in using RTOS. |
| Channel | Manual Entry | 0 | IIC channel |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Transfer rate for lower level driver. |

**Configurations for Connectivity > I2C Communication Device (rm_comms_i2c)**

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Communication Device (rm_comms_i2c).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comms_i2c_device0 | Module name. |
| Semaphore Timeout (RTOS only) | Value must be a non-negative integer less than or equal to 0xFFFFFFFF | 0xFFFFFFFF | Set timeout for blocking in using RTOS. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the I2C address mode. |
| Callback | Name must be a valid C symbol | comms_i2c_callback | A user callback function can be provided. |

**Pin Configuration**

This module uses SDA and SCL pins of I2C Master, SCI I2C, IICA Master and SAU I2C.

# Usage Notes

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.
  If an RTOS is used and blocking and bus lock is enabled, RM_COMMS_I2C_Write(), RM_COMMS_I2C_Read() and RM_COMMS_I2C_WriteRead() cannot be called in callback.

### Bus Initialization

The I2C communications interface expects a bus instance to be opened before opening any specific I2C comms device. The communications interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

# Examples

### Basic Example

This is a basic example of minimal use of I2C communications implementation in an application.

```
void rm_comms_i2c_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend          =
(rm_comms_i2c_bus_extended_cfg_t *) g_comms_i2c_cfg.p_extend;
 i2c_master_instance_t          * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
```

```
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_COMMS_I2C_Open(&g_comms_i2c_ctrl, &g_comms_i2c_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
       g_flag = 0;
 /* Send data to an I2C device. */
 RM_COMMS_I2C_Write(&g_comms_i2c_ctrl, &write_data, 1);
 while (0 == g_flag)
      {
 /* Wait callback */
      }
```

```
      g_flag = 0;
 /* Receive data from an I2C device. */
 RM_COMMS_I2C_Read(&g_comms_i2c_ctrl, &read_data, 1);
 while (0 == g_flag)
     {
 /* Wait callback */
     }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_comms_i2c_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_comms_i2c_instance_ctrl_t

| struct rm_comms_i2c_instance_ctrl_t | |
|---|---|
| Communications middleware control structure. | |
| **Data Fields** | |
| rm_comms_cfg_t const * | p_cfg |
| | middleware configuration. |
| | |
| rm_comms_i2c_bus_extended_cfg_t * | p_bus |
| | Bus using this device;. |
| | |
| void * | p_lower_level_cfg |
| | Used to reconfigure I2C driver. |
| | |
| uint32_t | open |
| | Open flag. |
| | |
| uint32_t | transfer_data_bytes |

| | |
|---|---|
| | Size of transfer data. |
| | |
| uint8_t * | p_transfer_data |
| | Pointer to transfer data buffer. |
| | |
| bool | smbus_operation |
| | SMBus operation on I2C bus. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |

### Function Documentation

#### ◆ RM_COMMS_I2C_Open()

fsp_err_t RM_COMMS_I2C_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the Communications Middle module. Implements rm_comms_api_t::open.

Example:

```
    err = RM_COMMS_I2C_Open(&g_comms_i2c_ctrl, &g_comms_i2c_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Communications Middle module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_COMMS_BUS_NOT_OPEN | I2C driver is not open. |

### ◆ RM_COMMS_I2C_Close()

fsp_err_t RM_COMMS_I2C_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified Communications Middle module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_I2C_Read()

fsp_err_t RM_COMMS_I2C_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the I2C device. Implements rm_comms_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_I2C_Write()

fsp_err_t RM_COMMS_I2C_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write from the I2C device. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_I2C_WriteRead()

fsp_err_t RM_COMMS_I2C_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs a write to, then a read from the I2C device. Implements rm_comms_api_t::writeRead.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## 5.2.6.6 I2C Master (r_iic_b_master)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_IIC_B_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_B_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_B_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_B_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IIC_B_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_IIC_B_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IIC_B_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_IIC_B_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl) |

### Detailed Description

I2C Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The I2C master on IIC/I3C HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_iic_b_master

The following build time configurations are defined in fsp_cfg/r_iic_b_master_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

### Configurations for Connectivity > I2C Master (r_iic_b_master)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_iic_b_master). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c_master0 | Module name. |
| Channel | Manual Entry | 0 | Specify the IIC channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and duty cycle are printed in a comment in the generated iic_b_master_extended_cfg_t structure. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000; Fast-mode plus: up to 1000000 |
| Rise Time (ns) | Value must be a non-negative integer | 120 | Set the rise time (tr) in nanoseconds. |
| Fall Time (ns) | Value must be a non-negative integer | 120 | Set the fall time (tf) in nanoseconds. |
| Duty Cycle (%) | Value must be an integer between 0 and 100 | 50 | Set the SCL duty cycle. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here. |
| Timeout Mode | • Short Mode<br>• Long Mode | Short Mode | Select the timeout mode to detect bus hang. |
| Timeout during SCL | • Enabled | Enabled | Select if the timeout |

| | | | |
|---|---|---|---|
| Low | • Disabled | | can occur when SCL is held low for a duration longer than what is set in the timeout mode. |
| Callback | Name must be a valid C symbol | g_iic_b_master0_callback | A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status. |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts. |

### Clock Configuration

The I3C peripheral module uses the IICCLK or PCLKD (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

## Usage Notes

### Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

### IIC Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
  - Fast-mode Plus (Fm+) : up to 1 Mbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested

rate) at the current IICCLK/PCLKD (based on the MCU) settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

### Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

### Restart

- IIC master can hold the the bus after an I2C transaction by issuing a repeated start condition.

### Limitations

- DTC should be disabled for data transmission on RA2E2 if MCU version is 1. Please refer to "MCUVER : MCU Version Register" description in the Flash Memory section of the relevant hardware manual for how to find MCU version.

## Examples

### Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_FAST,
    .slave         = I2C_SLAVE_EEPROM,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
```

```
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
```

```
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

*Note*

*The callback function from the first example applies to this example as well.*

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_2;
```

```
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
```

```
    }
 /* Send data to I2C slave on the same channel */
    err = R_IIC_B_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iic_b_master_clock_settings_t |
| struct | iic_b_master_instance_ctrl_t |
| struct | iic_b_master_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | iic_b_master_timeout_mode_t |
| enum | iic_b_master_timeout_scl_low_t |

## Data Structure Documentation

### ◆ iic_b_master_clock_settings_t

struct iic_b_master_clock_settings_t

| I2C clock settings | | |
|---|---|---|
| Data Fields | | |
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brh_value | High-level period of SCL clock. |
| uint8_t | brl_value | Low-level period of SCL clock. |
| uint8_t | sdod_value | |
| bool | sdodcs_value | |

### ◆ iic_b_master_instance_ctrl_t

| struct iic_b_master_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ iic_b_master_extended_cfg_t

| struct iic_b_master_extended_cfg_t | | |
|---|---|---|
| R_IIC_B extended configuration | | |
| Data Fields | | |
| iic_b_master_timeout_mode_t | timeout_mode | Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1. |
| iic_b_master_timeout_scl_low_t | timeout_scl_low | Allows timeouts to occur when SCL is held low. |
| iic_b_master_clock_settings_t | clock_settings | I2C Clock settings. |
| uint32_t | iic_clock_freq | I2C Clock frequency in Hz. |
| bool | smbus_operation | SMBus operation on I2C bus. |

## Enumeration Type Documentation

### ◆ iic_b_master_timeout_mode_t

| enum iic_b_master_timeout_mode_t | |
|---|---|
| I2C Timeout mode parameter definition | |
| Enumerator | |
| IIC_B_MASTER_TIMEOUT_MODE_LONG | Timeout Detection Time Select: Long Mode -> TMOS = 0. |
| IIC_B_MASTER_TIMEOUT_MODE_SHORT | Timeout Detection Time Select: Short Mode -> TMOS = 1. |

### ◆ iic_b_master_timeout_scl_low_t

| enum iic_b_master_timeout_scl_low_t | |
|---|---|
| Enumerator | |
| IIC_B_MASTER_TIMEOUT_SCL_LOW_DISABLED | Timeout detection during SCL low disabled. |
| IIC_B_MASTER_TIMEOUT_SCL_LOW_ENABLED | Timeout detection during SCL low enabled. |

## Function Documentation

### ◆ R_IIC_B_MASTER_Open()

fsp_err_t R_IIC_B_MASTER_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned |

### ◆ R_IIC_B_MASTER_Read()

fsp_err_t R_IIC_B_MASTER_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl, p_dest or bytes is NULL, or DTC is used for data transmission on RA2E2 Ver 1. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block. |

### ◆ R_IIC_B_MASTER_Write()

fsp_err_t R_IIC_B_MASTER_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block. |

### ◆ R_IIC_B_MASTER_Abort()

fsp_err_t R_IIC_B_MASTER_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel was reset successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

### ◆ R_IIC_B_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_B_MASTER_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | Pointer to control structure is NULL. |
| FSP_ERR_IN_USE | Another transfer was in-progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block. |

### ◆ R_IIC_B_MASTER_CallbackSet()

fsp_err_t R_IIC_B_MASTER_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_IIC_B_MASTER_StatusGet()

fsp_err_t R_IIC_B_MASTER_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_IIC_B_MASTER_Close()

| fsp_err_t R_IIC_B_MASTER_Close ( i2c_master_ctrl_t *const *p_api_ctrl*) |
|---|

Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

## 5.2.6.7 I2C Master (r_iic_master)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_IIC_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IIC_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IIC_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_IIC_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IIC_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_IIC_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The I2C master on IIC HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_iic_master

The following build time configurations are defined in fsp_cfg/r_iic_master_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

### Configurations for Connectivity > I2C Master (r_iic_master)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_iic_master). Non-secure callable guard functions can be generated for this module by right clicking

the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c_master0 | Module name. |
| Channel | Manual Entry | 0 | Specify the IIC channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and duty cycle are printed in a comment in the generated iic_master_extended_cfg_t structure. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000; Fast-mode plus: up to 1000000 |
| Rise Time (ns) | Value must be a non-negative integer | 120 | Set the rise time (tr) in nanoseconds. |
| Fall Time (ns) | Value must be a non-negative integer | 120 | Set the fall time (tf) in nanoseconds. |
| Duty Cycle (%) | Value must be an integer between 0 and 100 | 50 | Set the SCL duty cycle. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here. |

| | | | |
|---|---|---|---|
| Timeout Mode | • Short Mode<br>• Long Mode | Short Mode | Select the timeout mode to detect bus hang. |
| Timeout during SCL Low | • Enabled<br>• Disabled | Enabled | Select if the timeout can occur when SCL is held low for a duration longer than what is set in the timeout mode. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status. |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts. |

### Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

### IIC Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps

    - Fast-mode (Fm) : up to 400 kbps
    - Fast-mode Plus (Fm+) : up to 1 Mbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

### Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

### Restart

- IIC master can hold the the bus after an I2C transaction by issuing a repeated start condition.

# Examples

### Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_1;

i2c_master_cfg_t g_i2c_device_cfg_1 =

{

    .channel       = I2C_CHANNEL,

    .rate          = I2C_MASTER_RATE_FAST,

    .slave         = I2C_SLAVE_EEPROM,

    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,

    .p_callback    = i2c_callback,    // Callback

    .p_context    = &g_i2c_device_ctrl_1,
```

```
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
```

```
    {
        __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

*Note*

*The callback function from the first example applies to this example as well.*

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
```

```
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
```

```
    err = R_IIC_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iic_master_clock_settings_t |
| struct | iic_master_instance_ctrl_t |
| struct | iic_master_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | iic_master_timeout_mode_t |
| enum | iic_master_timeout_scl_low_t |

## Data Structure Documentation

### ◆ iic_master_clock_settings_t

| struct iic_master_clock_settings_t |
|---|
| I2C clock settings |
| Data Fields |

| | | |
|---|---|---|
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brh_value | High-level period of SCL clock. |
| uint8_t | brl_value | Low-level period of SCL clock. |
| uint8_t | sddl_value | |
| bool | dlcs_value | |

### ◆ iic_master_instance_ctrl_t

| struct iic_master_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ iic_master_extended_cfg_t

| struct iic_master_extended_cfg_t | | |
|---|---|---|
| R_IIC extended configuration | | |
| Data Fields | | |
| iic_master_timeout_mode_t | timeout_mode | Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1. |
| iic_master_timeout_scl_low_t | timeout_scl_low | Allows timeouts to occur when SCL is held low. |
| iic_master_clock_settings_t | clock_settings | I2C Clock settings. |
| bool | smbus_operation | SMBus operation on I2C bus. |

## Enumeration Type Documentation

### ◆ iic_master_timeout_mode_t

| enum iic_master_timeout_mode_t | |
|---|---|
| I2C Timeout mode parameter definition | |
| Enumerator | |
| IIC_MASTER_TIMEOUT_MODE_LONG | Timeout Detection Time Select: Long Mode -> TMOS = 0. |
| IIC_MASTER_TIMEOUT_MODE_SHORT | Timeout Detection Time Select: Short Mode -> TMOS = 1. |

### ◆ iic_master_timeout_scl_low_t

| enum iic_master_timeout_scl_low_t | |
|---|---|
| Enumerator | |
| IIC_MASTER_TIMEOUT_SCL_LOW_DISABLED | Timeout detection during SCL low disabled. |
| IIC_MASTER_TIMEOUT_SCL_LOW_ENABLED | Timeout detection during SCL low enabled. |

## Function Documentation

### ◆ R_IIC_MASTER_Open()

fsp_err_t R_IIC_MASTER_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned |

### ◆ R_IIC_MASTER_Read()

fsp_err_t R_IIC_MASTER_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl, p_dest or bytes is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_Write()

fsp_err_t R_IIC_MASTER_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_IN_USE | Bus busy condition. Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_Abort()

fsp_err_t R_IIC_MASTER_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel was reset successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

### ◆ R_IIC_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_MASTER_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | Pointer to control structure is NULL. |
| FSP_ERR_IN_USE | Another transfer was in-progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

### ◆ R_IIC_MASTER_CallbackSet()

fsp_err_t R_IIC_MASTER_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_IIC_MASTER_StatusGet()

fsp_err_t R_IIC_MASTER_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_IIC_MASTER_Close()

| fsp_err_t R_IIC_MASTER_Close ( i2c_master_ctrl_t *const *p_api_ctrl*) |
|---|
| Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

## 5.2.6.8 I2C Master (r_iica_master)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_IICA_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_IICA_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IICA_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_IICA_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IICA_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_IICA_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IICA_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_IICA_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the IICA peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The IICA master on IICA HAL module supports transactions with an IICA slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- IICA Master Read from a slave device.
- IICA Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.

# Configuration

### Build Time Configurations for r_iica_master

The following build time configurations are defined in fsp_cfg/r_iica_master_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |
| Enable Single Channel | MCU Specific Options | | Enable single channel to reduce code size if only one channel is to be configured for IICA. |

### Configurations for Connectivity > IICA Master (r_iica_master)

This module can be added to the Stacks tab via New Stack > Connectivity > IICA Master (r_iica_master).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid | g_iica_master0 | Module name. |

| | | | |
|---|---|---|---|
| | C symbol | | |
| Channel | Value must be a non-negative integer | 0 | Specify the IICA channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate is printed in a comment in the generated iica_master_extended_cfg_t structure. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000; Fast-mode plus: up to 1000000 |
| Signal Rising Time (us) | Must be a valid value | 0 | Set the SDA and SCL signal rising time in micro-seconds. |
| Signal Falling Time (us) | Must be a valid value | 0 | Set the SDA and SCL signal falling time in micro-seconds. |
| Duty Cycle (%) | Value must be an integer between 0 and 100 | 53 | Set SCL high duty cycle. |
| Digital Filter | • Enabled<br>• Disabled | Disabled | Configure digital filter. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here. |
| Slave Address | Value must be a non-negative integer | 0x00 | Specify the slave address. |
| Callback | Name must be a valid | iica_master_callback | A user callback |

| | | |
|---|---|---|
| | C symbol | function must be provided. This will be called from the interrupt service routine (ISR) upon IICA transaction completion reporting the transaction status. |
| Interrupt Priority Level | MCU Specific Options | Select end of IICA communication interrupt priority. |

### Clock Configuration

The IICA peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The IICA peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An IICA channel would consist of two pins - SDAA and SCLA for data/address and clock respectively.

## Usage Notes

### IICA Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
  - Fast-mode Plus (Fm+) : up to 1 Mbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Multiple Devices on the Bus

- A single IICA instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

### Restart

- IICA master can hold the bus after an I2C transaction by issuing a repeated start

condition.This will mimic a stop followed by start condition.

### Limitations

- DTC not supported.
- Interrupt request is always generated on the falling edge of the 9th clock cycle.
- Master operation in multi-master system is not supported
- Communication reservation function is not supported.

# Examples

### Basic Example

This is a basic example of minimal use of the r_iica_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iica_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_FAST,
    .slave         = I2C_SLAVE_EEPROM,
    .p_callback    = iica_master_callback, // Callback
    .p_context    = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void iica_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IICA_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IICA_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IICA_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
```

```
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IICA driver can be used to communicate with different slave devices which are on the same channel.

*Note*

*The callback function from the first example applies to this example as well.*

```
iica_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .p_callback    = iica_master_callback, // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IICA_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IICA_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_IICA_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IICA_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
```

```
    }
}
```

### Data Structures

| | |
|---|---|
| struct | iica_master_clock_settings_t |
| struct | iica_master_pin_settings_t |
| struct | iica_master_instance_ctrl_t |
| struct | iica_master_extended_cfg_t |

## Data Structure Documentation

### ◆ iica_master_clock_settings_t

| struct iica_master_clock_settings_t |
|---|
| IICA clock settings |

### ◆ iica_master_pin_settings_t

| struct iica_master_pin_settings_t | | |
|---|---|---|
| Configuration settings for IICA pins | | |
| Data Fields | | |
| bsp_io_port_pin_t | pin | The pin. |
| uint32_t | cfg | Configuration for the pin. |

### ◆ iica_master_instance_ctrl_t

| struct iica_master_instance_ctrl_t |
|---|
| IICA control structure. DO NOT INITIALIZE. |

### ◆ iica_master_extended_cfg_t

| struct iica_master_extended_cfg_t | | |
|---|---|---|
| R_IICA extended configuration | | |
| Data Fields | | |
| iica_master_clock_settings_t | clock_settings | |
| iica_master_pin_settings_t | sda_pin_settings | SDAA pin setting. |
| iica_master_pin_settings_t | scl_pin_settings | SCLAA pin setting. |

## Function Documentation

### ◆ R_IICA_MASTER_Open()

fsp_err_t R_IICA_MASTER_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the IICA device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Invalid IRQ number assigned |

### ◆ R_IICA_MASTER_Read()

fsp_err_t R_IICA_MASTER_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the IICA device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl, p_dest or bytes is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IICA_MASTER_Open to initialize the control block. |

### ◆ R_IICA_MASTER_Write()

fsp_err_t R_IICA_MASTER_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the IICA device. The caller will be notifieEd when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLET in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IICA_MASTER_Open to initialize the control block. |

### ◆ R_IICA_MASTER_Abort()

fsp_err_t R_IICA_MASTER_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IICA peripheral into ready state.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel was reset successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IICA_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

### ◆ R_IICA_MASTER_SlaveAddressSet()

fsp_err_t R_IICA_MASTER_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | Pointer to control structure is NULL. |
| FSP_ERR_IN_USE | Another transfer was in-progress. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IICA_MASTER_Open to initialize the control block. |

### ◆ R_IICA_MASTER_CallbackSet()

fsp_err_t R_IICA_MASTER_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_IICA_MASTER_StatusGet()

fsp_err_t R_IICA_MASTER_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_IICA_MASTER_Close()

fsp_err_t R_IICA_MASTER_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the IICA device. May power down IICA peripheral. This function will safely terminate any in-progress IICA transfers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_IICA_MASTER_Open to initialize the control block. |

***Note***

*A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.*

## 5.2.6.9 I2C Master (r_sau_i2c)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SAU_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_SAU_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SAU_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SAU_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_SAU_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SAU_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_SAU_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_I2C_Start (sau_i2c_instance_ctrl_t *const p_ctrl) |

fsp_err_t R_SAU_I2C_Stop (sau_i2c_instance_ctrl_t *const p_ctrl)

### Detailed Description

Driver for the SAU peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The Simple I2C master on SAU HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.

# Configuration

### Build Time Configurations for r_sau_i2c

The following build time configurations are defined in fsp_cfg/r_sau_i2c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable Critical Section | • Enabled<br>• Disabled | Disabled | Critical section needs to be enabled if multiple channels on the same SAU unit are configured for I2C |
| Manual Start-Stop | • Enabled<br>• Disabled | Disabled | If enabled, users need to manually call the R_SAU_I2C_Start/R_SAU_I2C_Stop functions to generate the I2C start and stop conditions. |
| Enable Single Channel | MCU Specific Options | | Enable single channel to reduce code size if |

| | | | |
|---|---|---|---|
| | | | only one channel is to be configured for SAU I2C. |
| I2C Restart | • Enable<br>• Disable | Enable | Select whether to include code for the I2C restart (repeated start) condition in the build. Set to 'Disable' to reduce code size when all I2C slaves used by all SAU instances do not use I2C restart. |
| DTC Support | • Enable<br>• Disable | Disable | Enable DTC support for the SAU I2C module. |

**Configurations for Connectivity > I2C Master (r_sau_i2c)**

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sau_i2c).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_i2c0 | Module name. |
| Channel | MCU Specific Options | | Select the SAU channel. |
| Operation clock | • CKm0<br>• CKm1 | CKm0 | Select the I2C operation clock. Use the Clocks tab to set the operation clock divider. |
| Slave Address | Value must be a non-negative integer | 0x00 | Specify the slave address. |
| Rate | • Standard<br>• Fast mode<br>• Fast mode plus | Standard | Select the I2C data rate.<br><br>If the requested rate cannot be achieved, adjust the operation clock frequency until the rate is achievable. The calculated rate is printed in a comment in the generated sau_i2c_extended_cfg_t structure. |
| Delay time (Microseconds) | Value must be a non-negative integer | 5 | Hold SDA (or SCK) for delay time to meet the I2C start (or stop) condition. Needs to be |

| | | | |
|---|---|---|---|
| | | | specified according to slave devices. |
| Callback | Name must be a valid C symbol | sau_i2c_master_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Transfer end interrupt priority | MCU Specific Options | | Select transfer end interrupt priority. This is set for TEI interrupt. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000; Fast-mode plus: up to 1000000 |

**Clock Configuration**

The SAU clock uses the system clock (ICLK) as its clock source.

A prescaler is applied to the ICLK in order to produce the operation clock frequency. The operation clock is used to generate the desired transfer period of the SAU module.

SAU operation clocks are shared among all channels within a SAU unit. Check the Hardware User's Manual for your MCU for available units and channels. SAU operation clock dividers are configurable in the Clocks tab.

The operation clock dividers are named SAU CKmn where m is the SAU unit, and n is the operation clock. For example, SAU CK01 applies to all SAU0 instances using CK1 as the operation clock (m=0, n=1).

**Pin Configuration**

The SAU I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

**Enabling Single Channel By User With The SAU I2C**

- The Common > Single Channel property is used for reducing code size when only 1 SAU channel is to be configured for SAU I2C.
- Single Channel is configurable and disabled by default in the driver code.
- Note: Not all SAU channels are available on all pin layouts. Check the Hardware User Manual for your device to confirm available function assignment for each SAU channel.

### Enabling Start/Stop Condition By User With The SAU I2C

- Manual triggering of I2C start/stop conditions is configurable and disabled by default.
- The delay time between the rising edge of SCL and the rising edge of SDA (stop condition) or the falling edge of SDA and the falling edge of SCL (start condition) is blocking. If blocking in the ISR for the length of the delay time is not suitable for the application, users can decide the appropriate calling time for start/stop conditions by enabling manual triggering and calling R_SAU_I2C_Start/R_SAU_I2C_Stop from the application context.
- When manual triggering is disabled, the driver will generate the stop condition after data transmission from the ISR context.

### Interrupt Configuration

- Transmit end interrupt (TEI) for the selected channel used must be enabled in the properties of the selected device.

### SAU I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and operating clock.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
  - Fast-mode Plus (Fm+) : up to 1 Mbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Selecting Operation Clock Frequency

The relationship between operation clock frequency and bitrate is: bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ] where:

- SDRmn.STCLK is an integer in the range [1, 127] for SAU I2C
- f_mck is the operation clock (SAU CKmn) frequency

By plugging in the minimum and maximum SDRmn.STCLK values, the range of bitrates for a given operation clock frequency can be obtained.

Note that due to STCLK being set as discrete integers, the actual bitrate may not be exact. The actual bitrate and percent errors can be calculated by the formulas:

```
actual_bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ]

percent_error = 100 * abs [(actual_bitrate - expected_bitrate) / expected_bitrate]
```

Using the fastest possible operation clock for the desired bitrate will result in the lowest deviation from the requested bitrate. Set the CKmn operation clock divider in the Clocks tab to select the desired operation clock frequency.

### Enabling DTC with the SAU I2C

- DTC transfer support is configurable and is disabled from the build by default. The SAU I2C driver uses a single DTC instance for transmission and reception.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- If DTC is not used by any SAU instance, set Common > DTC Support to disabled to reduce code size.
- For further details on DTC please refer Transfer (r_dtc)

### Multiple Channels Being Used In Same Unit

- When multiple channels on the same SAU unit are used for any serial function (I2C, UART, and/or SPI), then the Common > Enable Critical Section property needs to be set to enabled for all SAU serial drivers.
    - This is because some SAU registers are shared for SAU channels on the same unit. Therefore any read-modify-write operations to those registers must be protected by a critical section.

### I2C Restart Support

- I2C Restart (repeated start) is required by some, but not all slave devices.
- If no slave devices used by any SAU I2C instance require repeated start, then the Common > I2C Restart property can be set to disabled to reduce code size.
- When I2C Restart is disabled, even if restart=true is passed to the read or write APIs, the restart condition is not issued.

### Multiple Devices On The Bus

- A single SAU I2C instance can be used to communicate with multiple slave devices on the same channel by using the R_SAU_I2C_SlaveAddressSet API.

### Limitations

- Overrun error detection is not supported
- SAU I2C does not support clock stretching, I2C slave mode, or arbitration loss detection

## Examples

### Basic Example

This is a basic example of minimal use of the r_sau_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
```

```
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SAU_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SAU_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SAU_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Enabling Start/Stop Condition By User

This example demonstrates how to write code for SAU I2C communication when the call start/stop condtion by user is enabled.

```
void manual_trigger_condition_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SAU_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
```

```
    err = R_SAU_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
    err = R_SAU_I2C_Start(&g_i2c_device_ctrl_1);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
    err = R_SAU_I2C_Stop(&g_i2c_device_ctrl_1);
    assert(FSP_SUCCESS == err);
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SAU_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
    err = R_SAU_I2C_Start(&g_i2c_device_ctrl_1);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
    err = R_SAU_I2C_Stop(&g_i2c_device_ctrl_1);
    assert(FSP_SUCCESS == err);
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
```

```
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
    }
}
```

### Multiple Slave Devices On The Same Channel (Bus)

This example demonstrates how a single SAU I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SAU_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SAU_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
```

```
    {
        __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SAU_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2, I2C_SLAVE_DISPLAY_ADAPTER,
I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SAU_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | sau_i2c_clock_settings_t |
| struct | sau_i2c_instance_ctrl_t |
| struct | sau_i2c_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | sau_i2c_operation_clock_t |

## Data Structure Documentation

### ◆ sau_i2c_clock_settings_t

| struct sau_i2c_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| uint8_t | stclk | Bit rate register settings. |
| sau_i2c_operation_clock_t | operation_clock | I2C operating clock select. |

### ◆ sau_i2c_instance_ctrl_t

| struct sau_i2c_instance_ctrl_t | |
|---|---|
| I2C control structure. DO NOT INITIALIZE. | |
| **Data Fields** | |
| i2c_master_cfg_t const * | p_cfg |
| | Pointer to the configuration structure. |
| | |
| uint32_t | open |
| | Flag to determine if the device is open. |
| | |
| R_SAU0_Type * | p_reg |
| | Base register for this channel. |
| | |
| uint8_t * | p_buff |
| | Holds the data associated with the transfer. |
| | |
| uint32_t | total |
| | Holds the total number of data bytes to transfer. |
| | |
| uint32_t | loaded |
| | Tracks the number of data bytes written to the register. |
| | |
| volatile bool | read |
| | Holds the direction of the data byte transfer. |

| | |
|---|---|
| | |
| volatile bool | restart |
| | Holds whether or not the restart should be issued when done. |
| | |
| volatile bool | restarted |
| | Tracks whether or not a restart was issued during the previous transfer. |
| | |
| volatile bool | do_dummy_read |
| | Tracks whether a dummy read is issued on the first RX. |
| | |
| uint8_t | slave |
| | The address of the slave device. |
| | |

### ◆ sau_i2c_extended_cfg_t

| struct sau_i2c_extended_cfg_t | | |
|---|---|---|
| SAU I2C extended configuration | | |
| Data Fields | | |
| sau_i2c_clock_settings_t | clock_settings | I2C clock settings. |
| uint8_t | delay_time | The delay time of the slave device. |
| uint8_t | i2c_unit | The SAU unit corresponding to the selected channel. |

## Enumeration Type Documentation

### ◆ sau_i2c_operation_clock_t

| enum sau_i2c_operation_clock_t | |
|---|---|
| Operation clock | |
| Enumerator | |
| SAU_I2C_MASTER_OPERATION_CLOCK_CK0 | Operating clock select CK0. |
| SAU_I2C_MASTER_OPERATION_CLOCK_CK1 | Operating clock select CK1. |

## Function Documentation

### ◆ R_SAU_I2C_Open()

fsp_err_t R_SAU_I2C_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the SAU device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Clock rate requested is greater than 400KHz<br>5. Invalid IRQ number assigned |

### ◆ R_SAU_I2C_Read()

fsp_err_t R_SAU_I2C_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl, p_dest is NULL, bytes is 0. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SAU_I2C_Write()

fsp_err_t R_SAU_I2C_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. The write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl, p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SAU_I2C_Abort()

fsp_err_t R_SAU_I2C_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transaction was aborted without issue. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SAU_I2C_SlaveAddressSet()

fsp_err_t R_SAU_I2C_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address of the slave device.

This function is used to set the device address of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | p_ctrl or address is NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |
| FSP_ERR_IN_USE | An I2C Transaction is in progress. |

### ◆ R_SAU_I2C_CallbackSet()

fsp_err_t R_SAU_I2C_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback.

***Note***

*p_callback_memory is not used in this implementation and can be set to NULL.*

Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_SAU_I2C_StatusGet()

fsp_err_t R_SAU_I2C_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_SAU_I2C_Close()

fsp_err_t R_SAU_I2C_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not open. |

### ◆ R_SAU_I2C_Start()

fsp_err_t R_SAU_I2C_Start ( sau_i2c_instance_ctrl_t *const *p_ctrl*)

This function starts/restarts the IIC condition.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Instance control structure. |

### ◆ R_SAU_I2C_Stop()

fsp_err_t R_SAU_I2C_Stop ( sau_i2c_instance_ctrl_t *const *p_ctrl*)

This function stops the IIC condition.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Instance control structure. |

## 5.2.6.10 I2C Master (r_sci_b_i2c)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_SCI_B_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_B_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |

| | |
|---|---|
| fsp_err_t | R_SCI_B_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SCI_B_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_SCI_B_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_B_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_SCI_B_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the SCI_B peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The Simple I2C master on SCI_B HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_sci_b_i2c

The following build time configurations are defined in fsp_cfg/r_sci_b_i2c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

### Configurations for Connectivity > I2C Master (r_sci_b_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sci_b_i2c). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c0 | Module name. |
| Channel | Manual Entry | 0 | Select the SCI channel. |
| Slave Address | Value must be a hex value | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the address mode. |
| Rate | • Standard<br>• Fast-mode | Standard | Select the I2C data rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and SDA delay are printed in a comment in the generated sci_b_i2c_extended_cfg_t structure. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use |

| | | | |
|---|---|---|---|
| | | | the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000 |
| SDA Output Delay (nano seconds) | Must be a valid non-negative integer with maximum configurable value of 300 | 300 | Specify the SDA output delay in nanoseconds. |
| Noise Filter Clock Select | • The on-chip baud rate generator source clock divided by 1<br>• The on-chip baud rate generator source clock divided by 2<br>• The on-chip baud rate generator source clock divided by 4<br>• The on-chip baud rate generator source clock divided by 8 | The on-chip baud rate generator source clock divided by 1 | Select the on-chip baud rate generator source clock division setting for the digital noise filter |
| Clock Source | MCU Specific Options | | Select the clock source for the SCI I2C module. |
| Bit Rate Modulation | • Enable<br>• Disable | Enable | Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave. |
| Callback | Name must be a valid C symbol | sci_b_i2c_master_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |

| | | |
|---|---|---|
| Interrupt Priority Level | MCU Specific Options | Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts. |
| RX Interrupt Priority Level [Only used when DTC is enabled] | MCU Specific Options | Select the interrupt priority level. This is set for RXI only when DTC is enabled. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA6T2 | PCLKA |
| RA8D1 | PCLKA |
| RA8D2 | PCLKA |
| RA8E1 | PCLKA |
| RA8E2 | PCLKA |
| RA8M1 | PCLKA |
| RA8M2 | PCLKA |
| RA8P1 | PCLKA |
| RA8T1 | PCLKA |
| RA8T2 | PCLKA |

The actual I2C transfer rate can be derived from either SCISPICLK or the peripheral clock (PCLK)[1], and will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay. If the selected clock is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

*Note*

*1. See Figure 26.2 in the RA6T2 manual for more information.*

### Pin Configuration

The SCI_B I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to

different priority levels could result in improper operation.

### SCI_B I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Enabling DTC with the SCI_B I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI_B I2C driver provides two DTC instances for transmission and reception respectively.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

### Multiple Devices on the Bus

- A single SCI_B I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Restart

- SCI_B_I2C can hold the the bus after an I2C transaction by issuing a repeated start condition.

# Examples

### Basic Example

This is a basic example of minimal use of the r_sci_b_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
```

```
   err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Write some data to the transmit buffer */
for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
   {
      g_i2c_tx_buffer[i] = (uint8_t) i;
   }
/* Send data to I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers*/
while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
   {
      __BKPT(0);
   }
/* Read data back from the I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
   err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers*/
while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
```

```
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI_B I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
```

```
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_B_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | sci_b_i2c_clock_settings_t |
| struct | sci_b_i2c_instance_ctrl_t |
| struct | sci_b_i2c_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | sci_b_i2c_clock_source_t |

## Data Structure Documentation

### ◆ sci_b_i2c_clock_settings_t

| struct sci_b_i2c_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| bool | bitrate_modulation | Bit-rate Modulation Function enable or disable. |
| uint8_t | brr_value | Bit rate register settings. |
| uint8_t | clk_divisor_value | Clock Select settings. |
| uint8_t | mddr_value | Modulation Duty Register settings. |
| uint8_t | cycles_value | SDA Delay Output Cycles Select. |
| uint8_t | snfr_value | Noise Filter Setting Register value. |
| sci_b_i2c_clock_source_t | clock_source | Clock source (PCLK or SCISPICLK) |

### ◆ sci_b_i2c_instance_ctrl_t

| struct sci_b_i2c_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ sci_b_i2c_extended_cfg_t

| struct sci_b_i2c_extended_cfg_t | | |
|---|---|---|
| SCI I2C extended configuration | | |
| Data Fields | | |
| sci_b_i2c_clock_settings_t | clock_settings | I2C Clock settings. |

## Enumeration Type Documentation

### ◆ sci_b_i2c_clock_source_t

| enum sci_b_i2c_clock_source_t |
|---|
| SCI clock source |

## Function Documentation

### ◆ R_SCI_B_I2C_Open()

fsp_err_t R_SCI_B_I2C_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Clock rate requested is greater than 400KHz<br>5. Invalid IRQ number assigned |

### ◆ R_SCI_B_I2C_Read()

fsp_err_t R_SCI_B_I2C_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl, p_dest is NULL, bytes is 0. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_B_I2C_Write()

fsp_err_t R_SCI_B_I2C_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_ctrl, p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_B_I2C_Abort()

fsp_err_t R_SCI_B_I2C_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transaction was aborted without issue. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_B_I2C_SlaveAddressSet()

fsp_err_t R_SCI_B_I2C_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | p_ctrl or address is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |
| FSP_ERR_IN_USE | An I2C Transaction is in progress. |

### ◆ R_SCI_B_I2C_CallbackSet()

fsp_err_t R_SCI_B_I2C_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_B_I2C_StatusGet()

fsp_err_t R_SCI_B_I2C_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_SCI_B_I2C_Close()

fsp_err_t R_SCI_B_I2C_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

## 5.2.6.11 I2C Master (r_sci_i2c)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_SCI_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SCI_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart) |
| fsp_err_t | R_SCI_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, |

| | |
|---|---|
| | uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| fsp_err_t | R_SCI_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status) |
| fsp_err_t | R_SCI_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface.

# Overview

The Simple I2C master on SCI HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

# Configuration

### Build Time Configurations for r_sci_i2c

The following build time configurations are defined in fsp_cfg/r_sci_i2c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for |

| | | | |
|---|---|---|---|
| | | | both transmission and reception. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode. |

**Configurations for Connectivity > I2C Master (r_sci_i2c)**

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sci_i2c). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2c0 | Module name. |
| Channel | Manual Entry | 0 | Select the SCI channel. |
| Slave Address | Value must be a hex value | 0x00 | Specify the slave address. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the address mode. |
| Rate | • Standard<br>• Fast-mode | Standard | Select the I2C data rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and SDA delay are printed in a comment in the generated sci_i2c_extended_cfg_t structure. |
| Custom Rate (bps) | Value must be a non-negative integer | 0 | Set a custom bitrate (bps). Set to 0 to use the maximum bitrate for the selected mode.<br><br>Standard-mode: up to 100000; Fast-mode: up to 400000 |
| SDA Output Delay | Must be a valid non- | 300 | Specify the SDA output |

| | | | |
|---|---|---|---|
| (nano seconds) | negative integer with maximum configurable value of 300 | | delay in nanoseconds. |
| Noise filter setting | • Use clock signal divided by 1 with noise filter<br>• Use clock signal divided by 2 with noise filter<br>• Use clock signal divided by 4 with noise filter<br>• Use clock signal divided by 8 with noise filter | Use clock signal divided by 1 with noise filter | Select the sampling clock for the digital noise filter |
| Bit Rate Modulation | • Enable<br>• Disable | Enable | Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave. |
| Callback | Name must be a valid C symbol | sci_i2c_master_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Interrupt Priority Level | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts. |
| RX Interrupt Priority Level [Only used when DTC is enabled] | MCU Specific Options | | Select the interrupt priority level. This is set for RXI only when DTC is enabled. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |

| | |
|---|---|
| RA2A2 | PCLKB |
| RA2E1 | PCLKB |
| RA2E2 | PCLKB |
| RA2E3 | PCLKB |
| RA2L1 | PCLKB |
| RA2L2 | PCLKB |
| RA2T1 | PCLKB |
| RA4C1 | PCLKA |
| RA4E1 | PCLKA |
| RA4E2 | PCLKA |
| RA4L1 | PCLKA |
| RA4M1 | PCLKA |
| RA4M2 | PCLKA |
| RA4M3 | PCLKA |
| RA4T1 | PCLKA |
| RA4W1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T1 | PCLKA |
| RA6T3 | PCLKA |

The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay. If the PCLK is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The SCI I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

### SCI I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay.

  When the Custom Rate setting is set to 0 the bitrate is fixed to the maximum values shown below. Otherwise, the supplied value is used to generate bitrate settings.

  - Standard-mode (Sm) : up to 100 kbps
  - Fast-mode (Fm) : up to 400 kbps
- The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

### Enabling DTC with the SCI I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI I2C driver provides two DTC instances for transmission and reception respectively.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Transfer (r_dtc)

### Multiple Devices on the Bus

- A single SCI I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

### Restart

- SCI I2C master can hold the the bus after an I2C transaction by issuing Restart. This will mimic a stop followed by start condition.

## Examples

### Basic Example

This is a basic example of minimal use of the r_sci_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
```

```
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the receive buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2, I2C_SLAVE_DISPLAY_ADAPTER,
I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | sci_i2c_clock_settings_t |
| struct | sci_i2c_instance_ctrl_t |
| struct | sci_i2c_extended_cfg_t |

## Data Structure Documentation

### ◆ sci_i2c_clock_settings_t

| struct sci_i2c_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| bool | bitrate_modulation | Bit-rate Modulation Function enable or disable. |
| uint8_t | brr_value | Bit rate register settings. |
| uint8_t | clk_divisor_value | Clock Select settings. |
| uint8_t | mddr_value | Modulation Duty Register settings. |
| uint8_t | cycles_value | SDA Delay Output Cycles Select. |
| uint8_t | snfr_value | Noise Filter Setting Register value. |

### ◆ sci_i2c_instance_ctrl_t

| struct sci_i2c_instance_ctrl_t |
|---|
| I2C control structure. DO NOT INITIALIZE. |

### ◆ sci_i2c_extended_cfg_t

| struct sci_i2c_extended_cfg_t | | |
|---|---|---|
| SCI I2C extended configuration | | |
| Data Fields | | |
| sci_i2c_clock_settings_t | clock_settings | I2C Clock settings. |

## Function Documentation

### ◆ R_SCI_I2C_Open()

fsp_err_t R_SCI_I2C_Open ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg* )

Opens the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Requested clock rate was set exactly. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Clock rate requested is greater than 400KHz<br>5. Invalid IRQ number assigned |

### ◆ R_SCI_I2C_Read()

fsp_err_t R_SCI_I2C_Read ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart* )

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl, p_dest is NULL, bytes is 0. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_Write()

fsp_err_t R_SCI_I2C_Write ( i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart* )

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_ctrl, p_src is NULL. |
| FSP_ERR_INVALID_SIZE | Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_Abort()

fsp_err_t R_SCI_I2C_Abort ( i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transaction was aborted without issue. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

### ◆ R_SCI_I2C_SlaveAddressSet()

fsp_err_t R_SCI_I2C_SlaveAddressSet ( i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode* )

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Address of the slave is set correctly. |
| FSP_ERR_ASSERTION | p_ctrl or address is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |
| FSP_ERR_IN_USE | An I2C Transaction is in progress. |

### ◆ R_SCI_I2C_CallbackSet()

fsp_err_t R_SCI_I2C_CallbackSet ( i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_I2C_StatusGet()

fsp_err_t R_SCI_I2C_StatusGet ( i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status* )

Provides driver status.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ R_SCI_I2C_Close()

fsp_err_t R_SCI_I2C_Close ( i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed without issue. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Device was not even opened. |

## 5.2.6.12 I2C Slave (r_iic_b_slave)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| fsp_err_t | R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory) |

fsp_err_t R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl)

### Detailed Description

Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Slave Interface.

# Overview

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.
- Additional build-time features:
  - Optional (build time) DTC support for read and write respectively.

# Configuration

### Build Time Configurations for r_iic_b_slave

The following build time configurations are defined in fsp_cfg/r_iic_b_slave_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |

### Configurations for Connectivity > I2C Slave (r_iic_b_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Slave (r_iic_b_slave). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Interrupt Priority Level | | | |
| Transmit, Receive, and Transmit End | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, and TEI interrupts. |
| Error | MCU Specific Options | | Select the interrupt priority level. This is set for ERI interrupt. |
| Name | Name must be a valid C symbol | g_i2c_slave0 | Module name. |
| Channel | Manual Entry | 0 | Specify the IIC channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the delay for the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated delay is printed in a comment in the generated iic_b_slave_extended_cfg_t structure. |
| Internal Reference Clock | • I2C Clock / 1<br>• I2C Clock / 2<br>• I2C Clock / 4<br>• I2C Clock / 8<br>• I2C Clock / 16<br>• I2C Clock / 32<br>• I2C Clock / 64<br>• I2C Clock / 128 | I2C Clock / 1 | Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the clock frequency of the noise filter samples. I2C Clock can be either IICCLK or PCLKD based on the MCU. |
| Digital Noise Filter Stage Select | Refer to the RA Configuration tool for available options. | 3-stage filter | Select the number of digital filter stages for IIC Slave. |
| Slave Address | Value must be non-negative | 0x08 | Specify the slave address. |
| General Call | • Enabled<br>• Disabled | Disabled | Allows the slave to respond to general call address: 0x00. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. |

| | | | |
|---|---|---|---|
| Clock Stretching | • Enabled<br>• Disabled | Disabled | Configure Clock Stretching. |
| Callback | Name must be a valid C symbol | g_iic_b_slave0_callback | A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status. |

### Clock Configuration

The IIC/I3C peripheral module uses the IICCLK or PCLKD (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

### Pin Configuration

The IIC/I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.
- The interrupt priority of ERI can be set higher than or equal to the interrupt priorities of RXI, TXI and TEI.
  *Note*
  *: During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received, a higher priority ERI will ensure that the ongoing transaction is completed (by accepting the Stop/Restart condition from the master) before the next transaction is initiated.*
  *: To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.*

### Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
  - An I2C Master initiates a general call by passing 0x00 as slave address: I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes

transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

| IIC Slave Callback Event | IIC Slave API expected to be called |
|---|---|
| I2C_SLAVE_EVENT_ABORTED | Handle event based on application |
| I2C_SLAVE_EVENT_RX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_TX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_RX_REQUEST | R_IIC_B_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_REQUEST | R_IIC_B_SLAVE_Write API |
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | R_IIC_B_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | R_IIC_B_SLAVE_Write API |
| I2C_SLAVE_EVENT_GENERAL_CALL | R_IIC_B_SLAVE_Read |

- If parameter checking is enabled and R_IIC_B_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_B_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
    - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.
    - Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

### IIC Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current I2C Clock (IICCLK or PCLKD based on the MCU) settings is calculated and used.

### Limitations

- When 'Clock Stretching' configuration is enabled, the receive operation will not utilize the double buffer arrangement in hardware for a continuous read. This means that the read operation would happen in single byte units such that the active master would send the next byte only when the slave has read the current byte of data.
- When DTC is enabled, the clock frequency supply for DTC must be configured greater than the "Internal Reference Clock" of the slave.

# Examples

### Basic Example

This is a basic example of minimal use of the R_IIC_B_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_master_ctrl;
i2c_master_cfg_t g_i2c_master_cfg =
{
    .channel       = I2C_MASTER_CHANNEL_2,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_master_callback, // Callback
    .p_context    = &g_i2c_master_ctrl,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
iic_b_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
```

```
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_B_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
       err = R_IIC_B_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
```

```
 */
/* Initialize the IIC Slave module */
   err = R_IIC_B_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Initialize the IIC Master module */
   err = R_IIC_B_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
   assert(FSP_SUCCESS == err);
/* Write some data to the transmit buffer */
for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
   {
      g_i2c_master_tx_buffer[i] = (uint8_t) i;
   }
/* Send data to I2C slave */
   g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
   g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
   err = R_IIC_B_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers
 * The Slave Callback will call the R_IIC_B_SLAVE_Read API to service the Master
Write Request.
 */
while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
      (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
   {
      __BKPT(0);
   }
```

```
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

### Data Structures

| | |
|---|---|
| struct | iic_b_slave_clock_settings_t |
| struct | iic_b_slave_extended_cfg_t |

## Data Structure Documentation

### ◆ iic_b_slave_clock_settings_t

| struct iic_b_slave_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brl_value | Low-level period of SCL clock. |
| uint8_t | digital_filter_stages | Number of digital filter stages based on brl_value. |

### ◆ iic_b_slave_extended_cfg_t

| struct iic_b_slave_extended_cfg_t | | |
|---|---|---|
| R_IIC_SLAVE extended configuration | | |
| Data Fields | | |
| iic_b_slave_clock_settings_t | clock_settings | I2C Clock settings. |

## Function Documentation

### ◆ R_IIC_B_SLAVE_Open()

fsp_err_t R_IIC_B_SLAVE_Open ( i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg* )

Opens the I2C slave device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | I2C slave device opened successfully. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Error interrupt priority is lower than Transmit, Receive and Transmit End interrupt priority |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned<br>6. transfer instance in p_cfg is NULL when DTC support enabled |

### ◆ R_IIC_B_SLAVE_Read()

fsp_err_t R_IIC_B_SLAVE_Read ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue |
| FSP_ERR_ASSERTION | p_api_ctrl, bytes or p_dest is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |
| FSP_ERR_INVALID_SIZE | Invalid size when reading data via DTC. |

### ◆ R_IIC_B_SLAVE_Write()

fsp_err_t R_IIC_B_SLAVE_Write ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |
| FSP_ERR_INVALID_SIZE | Invalid size when writing data via DTC. |

### ◆ R_IIC_B_SLAVE_CallbackSet()

fsp_err_t R_IIC_B_SLAVE_CallbackSet ( i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_IIC_B_SLAVE_Close()

| fsp_err_t R_IIC_B_SLAVE_Close ( i2c_slave_ctrl_t *const *p_api_ctrl*) |
|---|
| Closes the I2C device.<br><br>**Return values**<br>FSP_SUCCESS — Device closed successfully.<br>FSP_ERR_NOT_OPEN — Device not opened.<br>FSP_ERR_ASSERTION — p_api_ctrl is NULL. |

## 5.2.6.13 I2C Slave (r_iic_slave)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_IIC_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| fsp_err_t | R_IIC_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_IIC_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| fsp_err_t | R_IIC_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IIC_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface.

# Overview

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.

- Write data which is read by master device.
- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.
- Additional build-time features:
    - Optional (build time) DTC support for read and write respectively.

# Configuration

### Build Time Configurations for r_iic_slave

The following build time configurations are defined in fsp_cfg/r_iic_slave_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC on Transmission and Reception | • Enabled<br>• Disabled | Disabled | If enabled, DTC instances will be included in the build for both transmission and reception. |

### Configurations for Connectivity > I2C Slave (r_iic_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Slave (r_iic_slave). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupt Priority Level | | | |
| Transmit, Receive, and Transmit End | MCU Specific Options | | Select the interrupt priority level. This is set for TXI, RXI, and TEI interrupts. |
| Error | MCU Specific Options | | Select the interrupt priority level. This is set for ERI interrupt. |
| Name | Name must be a valid C symbol | g_i2c_slave0 | Module name. |
| Channel | Manual Entry | 0 | Specify the IIC channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the delay for the |

| | | | |
|---|---|---|---|
| | | | requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated delay is printed in a comment in the generated iic_slave_extended_cfg_t structure. |
| Internal Reference Clock | • PCLKB / 1<br>• PCLKB / 2<br>• PCLKB / 4<br>• PCLKB / 8<br>• PCLKB / 16<br>• PCLKB / 32<br>• PCLKB / 64<br>• PCLKB / 128 | PCLKB / 1 | Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the clock frequency of the noise filter samples. |
| Digital Noise Filter Stage Select | • Disabled<br>• Single-stage filter<br>• 2-stage filter<br>• 3-stage filter<br>• 4-stage filter | 3-stage filter | Select the number of digital filter stages for IIC Slave. |
| Slave Address | Value must be non-negative | 0x08 | Specify the slave address. |
| General Call | • Enabled<br>• Disabled | Disabled | Allows the slave to respond to general call address: 0x00. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. |
| Clock Stretching | • Enabled<br>• Disabled | Disabled | Configure Clock Stretching. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status. |

### Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected transfer rate cannot be achieved, an error will be returned.

### Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.
- The interrupt priority of ERI can be set higher than or equal to the interrupt priorities of RXI, TXI and TEI.
  *Note*
  *: During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received, a higher priority ERI will ensure that the ongoing transaction is completed (by accepting the Stop/Restart condition from the master) before the next transaction is initiated.*
  *: To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.*

### Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception:
    I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed:
    I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data:
    I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
  - An I2C Master initiates a general call by passing 0x00 as slave address:
    I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

| IIC Slave Callback Event | IIC Slave API expected to be called |
|---|---|
| I2C_SLAVE_EVENT_ABORTED | Handle event based on application |
| I2C_SLAVE_EVENT_RX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_TX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_RX_REQUEST | R_IIC_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_REQUEST | R_IIC_SLAVE_Write API |

| | |
|---|---|
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | R_IIC_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | R_IIC_SLAVE_Write API |
| I2C_SLAVE_EVENT_GENERAL_CALL | R_IIC_SLAVE_Read |

- If parameter checking is enabled and R_IIC_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
  - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.
  - Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

### IIC Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.

### Limitations

- When 'Clock Stretching' configuration is enabled, the receive operation will not utilize the double buffer arrangement in hardware for a continuous read. This means that the read operation would happen in single byte units such that the active master would send the next byte only when the slave has read the current byte of data.
- When DTC is enabled, the clock frequency supply for DTC must be configured greater than the "Internal Reference Clock" of the slave.

## Examples

### Basic Example

This is a basic example of minimal use of the R_IIC_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_master_ctrl;

i2c_master_cfg_t g_i2c_master_cfg =

{

    .channel       = I2C_MASTER_CHANNEL_2,

    .rate          = I2C_MASTER_RATE_STANDARD,

    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,

    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,

    .p_callback    = i2c_master_callback, // Callback

    .p_context    = &g_i2c_master_ctrl,
```

```
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
iic_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
```

```
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
       err = R_IIC_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
  */
 /* Initialize the IIC Slave module */
    err = R_IIC_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the IIC Master module */
    err = R_IIC_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_master_tx_buffer[i] = (uint8_t) i;
```

```
    }
 /* Send data to I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    err = R_IIC_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Read API to service the Master Write
Request.
  */
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
```

```
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iic_slave_clock_settings_t |
| struct | iic_slave_extended_cfg_t |

## Data Structure Documentation

### ◆ iic_slave_clock_settings_t

| struct iic_slave_clock_settings_t | | |
|---|---|---|
| I2C clock settings | | |
| Data Fields | | |
| uint8_t | cks_value | Internal Reference Clock Select. |
| uint8_t | brl_value | Low-level period of SCL clock. |
| uint8_t | digital_filter_stages | Number of digital filter stages based on brl_value. |

### ◆ iic_slave_extended_cfg_t

| struct iic_slave_extended_cfg_t |
|---|
| R_IIC_SLAVE extended configuration |

| Data Fields | | |
|---|---|---|
| iic_slave_clock_settings_t | clock_settings | I2C Clock settings. |

### Function Documentation

#### ◆ R_IIC_SLAVE_Open()

fsp_err_t R_IIC_SLAVE_Open ( i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg* )

Opens the I2C slave device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | I2C slave device opened successfully. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Error interrupt priority is lower than Transmit, Receive and Transmit End interrupt priority |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Set the rate to fast mode plus on a channel which does not support it.<br>5. Invalid IRQ number assigned<br>6. transfer instance in p_cfg is NULL when DTC support enabled |

### ◆ R_IIC_SLAVE_Read()

fsp_err_t R_IIC_SLAVE_Read ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue |
| FSP_ERR_ASSERTION | p_api_ctrl, bytes or p_dest is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |
| FSP_ERR_INVALID_SIZE | Invalid size when reading data via DTC. |

### ◆ R_IIC_SLAVE_Write()

fsp_err_t R_IIC_SLAVE_Write ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |
| FSP_ERR_INVALID_SIZE | Invalid size when writing data via DTC. |

### ◆ R_IIC_SLAVE_CallbackSet()

fsp_err_t R_IIC_SLAVE_CallbackSet ( i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_IIC_SLAVE_Close()

fsp_err_t R_IIC_SLAVE_Close ( i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the I2C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_NOT_OPEN | Device not opened. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |

## 5.2.6.14 I2C Slave (r_iica_slave)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_IICA_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| fsp_err_t | R_IICA_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_IICA_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes) |

| | |
|---|---|
| fsp_err_t | R_IICA_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IICA_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the IICA peripheral on RA MCUs. This module implements the I2C Slave Interface.

# Overview

### Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- IICA Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.

# Configuration

### Build Time Configurations for r_iica_slave

The following build time configurations are defined in fsp_cfg/r_iica_slave_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| 10-bit slave addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support 10-bit slave addressing mode. |
| General call addressing | • Enabled<br>• Disabled | Disabled | If enabled, the driver will support general call slave addressing mode along with the non-general call slave addressing mode. |
| Enable Single Channel | MCU Specific Options | | Enable single channel to reduce code size if only one channel is to be configured for IICA. |

### Configurations for Connectivity > IICA Slave (r_iica_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > IICA Slave (r_iica_slave).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_iica_slave0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the IICA channel. |
| Rate | • Standard<br>• Fast-mode<br>• Fast-mode plus | Standard | Select the transfer rate.<br><br>If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate is printed in a comment in the generated iica_slave_extended_cfg_t structure. |
| Internal Reference Clock | • PCLKB / 1<br>• PCLKB / 2 | PCLKB / 1 | Select the internal reference clock for IICA slave. |
| Signal Rising Time (us) | Must be a valid value | 0 | Set the SDA and SCL signal rising time in micro-seconds. |
| Signal Falling Time (us) | Must be a valid value | 0 | Set the SDA and SCL signal falling time in micro-seconds. |
| Duty Cycle (%) | Value must be an integer between 0 and 100 | 53 | Set SCL high duty cycle. |
| Digital Filter | • Enabled<br>• Disabled | Disabled | Configure digital filter. |
| Address Mode | • 7-Bit<br>• 10-Bit | 7-Bit | Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here. |
| Slave Address | Value must be non-negative | 0x08 | Specify the slave address. |

| | | | |
|---|---|---|---|
| Callback | Name must be a valid C symbol | iica_slave_callback | A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report IICA Slave transaction events and status. |
| Interrupt Priority Level | MCU Specific Options | | Select end of IICA communication interrupt priority. |

### Clock Configuration

The IICA peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected transfer rate cannot be achieved, an error will be returned.

### Pin Configuration

The IICA peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An IICA channel would consist of two pins - SDAA and SCLA for data/address and clock respectively.

# Usage Notes

### Interrupt Configuration

- IICA0 communication interrupt must be enabled in the properties of the selected device.
  *Note*
  *: During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received.*
  *: To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.*

### Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IICA slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

| | |
|---|---|

| IICA Slave Callback Event | IICA Slave API expected to be called |
|---|---|
| I2C_SLAVE_EVENT_ABORTED | Handle event based on application |
| I2C_SLAVE_EVENT_RX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_TX_COMPLETE | Handle event based on application |
| I2C_SLAVE_EVENT_RX_REQUEST | R_IICA_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_REQUEST | R_IICA_SLAVE_Write API |
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | R_IICA_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | R_IICA_SLAVE_Write API |

### IICA Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.

### Limitations

- DTC doesn't support IICA
- Extension code is not supported

## Examples

### Basic Example

This is a basic example of minimal use of the R_IICA_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
sau_i2c_instance_ctrl_t g_i2c_master_ctrl;
i2c_master_cfg_t g_i2c_master_cfg =
{
    .channel       = I2C_MASTER_CHANNEL_20,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,
    .p_callback    = iica_master_callback, // Callback
    .p_context    = &g_i2c_master_ctrl,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
```

```
iica_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .p_callback = iica_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void iica_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void iica_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IICA_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
```

```
       err = R_IICA_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Initialize the IIC Slave module */
    err = R_IICA_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the IIC Master module */
    err = R_SAU_I2C_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_master_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    err = R_SAU_I2C_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IICA_SLAVE_Read API to service the Master
```

```
Write Request.
  */
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SAU_I2C_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IICA_SLAVE_Write API to service the Master
Read Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
```

```
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iica_slave_clock_settings_t |
| struct | iica_slave_pin_settings_t |
| struct | iica_slave_extended_cfg_t |

## Data Structure Documentation

### ◆ iica_slave_clock_settings_t

| struct iica_slave_clock_settings_t |
|---|
| IICA clock settings |

### ◆ iica_slave_pin_settings_t

| struct iica_slave_pin_settings_t | | |
|---|---|---|
| Configuration settings for IICA pins | | |
| Data Fields | | |
| bsp_io_port_pin_t | pin | The pin. |
| uint32_t | cfg | Configuration for the pin. |

### ◆ iica_slave_extended_cfg_t

| struct iica_slave_extended_cfg_t | | |
|---|---|---|
| R_IICA_SLAVE extended configuration | | |
| Data Fields | | |
| iica_slave_clock_settings_t | clock_settings | |
| iica_slave_pin_settings_t | sda_pin_settings | SDAA pin setting. |
| iica_slave_pin_settings_t | scl_pin_settings | SCLAA pin setting. |

## Function Documentation

### ◆ R_IICA_SLAVE_Open()

fsp_err_t R_IICA_SLAVE_Open ( i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg* )

Opens the IICA slave device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IICA slave device opened successfully. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel is not available on this MCU. |
| FSP_ERR_ASSERTION | Parameter check failure due to one or more reasons below:<br>1. p_api_ctrl or p_cfg is NULL.<br>2. extended parameter is NULL.<br>3. Callback parameter is NULL.<br>4. Invalid IRQ number assigned |

### ◆ R_IICA_SLAVE_Read()

fsp_err_t R_IICA_SLAVE_Read ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the IICA Master device.

This function will fail if there is already an in-progress IICA transfer on the associated channel. Otherwise, the IICA slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue |
| FSP_ERR_ASSERTION | p_api_ctrl, bytes or p_dest is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |

### ◆ R_IICA_SLAVE_Write()

fsp_err_t R_IICA_SLAVE_Write ( i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the IICA Master device.

This function will fail if there is already an in-progress IICA transfer on the associated channel. Otherwise, the IICA slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function executed without issue. |
| FSP_ERR_ASSERTION | p_api_ctrl or p_src is NULL. |
| FSP_ERR_IN_USE | Another transfer was in progress. |
| FSP_ERR_NOT_OPEN | Device is not open. |

### ◆ R_IICA_SLAVE_CallbackSet()

fsp_err_t R_IICA_SLAVE_CallbackSet ( i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_IICA_SLAVE_Close()

fsp_err_t R_IICA_SLAVE_Close ( i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the IICA device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_NOT_OPEN | Device not opened. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |

## 5.2.6.15 I2S (r_ssi)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SSI_Open (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg) |
| fsp_err_t | R_SSI_Write (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SSI_Read (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SSI_WriteRead (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SSI_Stop (i2s_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SSI_Mute (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable) |
| fsp_err_t | R_SSI_StatusGet (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status) |
| fsp_err_t | R_SSI_Close (i2s_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SSI_CallbackSet (i2s_ctrl_t *const p_api_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context, i2s_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface.

# Overview

### Features

The SSI module supports the following features:

- Transmission and reception of uncompressed audio data using the standard I2S protocol in master and slave modes
- Full-duplex I2S communication (channel 0 only)
- Integration with the DTC transfer module
- Internal connection to GPT timer output to generate the audio clock
- Callback function notification when all data is loaded into the SSI FIFO

# Configuration

### Build Time Configurations for r_ssi

The following build time configurations are defined in fsp_cfg/r_ssi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC Support | • Enabled<br>• Disabled | Enabled | If code for DTC transfer support is included in the build. |

### Configurations for Connectivity > I2S (r_ssi)

This module can be added to the Stacks tab via New Stack > Connectivity > I2S (r_ssi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i2s0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the I2S channel. |
| Operating Mode (Master/Slave) | • Master Mode<br>• Slave Mode | Master Mode | Select if the MCU is I2S master or slave. |
| Bit Depth | • 8 Bits<br>• 16 Bits<br>• 18 Bits<br>• 20 Bits<br>• 22 Bits<br>• 24 Bits<br>• 32 Bits | 16 Bits | Select the bit depth of one sample of audio data. |

| | | | |
|---|---|---|---|
| Word Length | • 8 Bits<br>• 16 Bits<br>• 24 Bits<br>• 32 Bits<br>• 48 Bits<br>• 64 Bits<br>• 128 Bits<br>• 256 Bits | 16 Bits | Select the word length of audio data. Must be at least as large as Data bits. |
| WS Continue Mode | • Enabled<br>• Disabled | Disabled | Enable WS continue mode to output the word select (WS) pin even when transmission is idle. |
| Bit Clock Source(available only in Master mode) | MCU Specific Options | | Select External AUDIO_CLK for external signal to AUDIO_CLK input pin or Internal AUDIO_CLK for internal connection to MCU specific GPT channel. Please refer to the hardware manual for which GPT channel is connected to the internal signal |
| Bit Clock Divider(available only in Master mode) | Refer to the RA Configuration tool for available options. | Audio Clock / 1 | Select divider used to generate bit clock from audio clock. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR). |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Idle/Error Interrupt Priority | MCU Specific Options | | Select the Idle/Error interrupt priority. |

### Clock Configuration

The SSI peripheral runs on PCLKB. The PCLKB frequency can be configured on the **Clocks** tab of the RA Configuration editor. The SSI audio clock can optionally be supplied from an external source through the AUDIO_CLK pin in master mode.

### Pin Configuration

The SSI uses the following pins:

- AUDIO_CLK (optional, master mode only): The AUDIO_CLK pin is used to supply the audio clock from an external source.
- SSIBCKn: Bit clock pin for channel n
- SSILRCKn/SSIFSn: Channel selection pin for channel n
- SSIRXD0: Reception pin for channel 0
- SSITXD0: Transmission pin for channel 0
- SSIDATA1: Transmission or reception pin for channel 1

# Usage Notes

### SSI Frames

An SSI frame is 2 samples worth of data. The frame boundary (end of previous frame, start of next frame) is on the falling edge of the SSILRCKn signal.

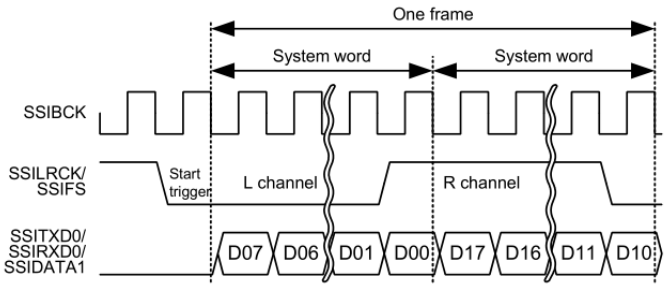



Figure 221: SSI Frame Diagram (8-bit word, 8-bit samples)

*Note*

*If the word length is longer than the sample bit depth, padding bits (0) will be added after the sample.*

### Audio Data

Only uncompressed PCM data is supported.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

| Bit Depth | Array Data Type | Required Alignment | Required Length (bytes) |
|---|---|---|---|
| 8 Bits | 8-bit integer | 1 byte alignment | Multiple of 2 |
| 16 Bits | 16-bit integer | 2 byte alignment | Multiple of 4 |
| 18 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 20 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 22 Bits | 32-bit integer, right justified | 4 byte alignment | Multiple of 8 |
| 24 Bits | 32-bit integer, right | 4 byte alignment | Multiple of 8 |

| | | | |
|---|---|---|---|
| | justified | | |
| 32 Bits | 32-bit integer | 4 byte alignment | Multiple of 8 |

*Note*

*The length of the array must be a multiple of 2 when the data type is the recommended data type. The 2 represents the frame size (left and right channel) of I2S communication. The SSIE peripheral does not support odd read/write lengths in I2S mode.*

### Audio Clock

The audio clock is only required for master mode.

### Audio Clock Frequency

The bit clock frequency is the product of the sampling frequency and channels and bits per system word:

bit_clock (Hz) = sampling_frequency (Hz) * channels * system_word_bits

I2S data always has 2 channels.

For example, the bit clock for transmitting 2 channels of 16-bit data (using a 16-bit system word) at 44100 Hz would be:

44100 * 2 * 16 = 1,411,200 Hz

The audio clock frequency is used to generate the bit clock frequency. It must be a multiple of the bit clock frequency. Refer to the Bit Clock Divider configuration for divider options. The input audio clock frequency must be:

audio_clock (Hz) = desired_bit_clock (Hz) * bit_clock_divider

To get a bit clock of 1.4 MHz from an audio clock of 2.8 MHz, select the divider Audio Clock / 2.

### Audio Clock Source

The audio clock source can come from:

- An external source input to the AUDIO_CLK pin
- An internal connection to the GPT timer output

*Note*

*When using the internal GPT timer output, Pin Output Support must be Enabled, and GTIOCA Output Enabled must be True.*
*See the SSIE section in the MCU hardware manual for information about which GPT channel may be used.*

### Limitations

Developers should be aware of the following limitations when using the SSI:

- When using channel 1, full duplex communication is not possible. Only tranmission or reception is possible.
- SSI must go idle before changing the communication mode (between read only, write only, and full duplex)

# Examples

### Basic Example

This is a basic example of minimal use of the SSI in an application.

```
#define SSI_EXAMPLE_SAMPLES_TO_TRANSFER (1024)
#define SSI_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t g_src[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
int16_t g_dest[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
void ssi_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t t = 0; t < SSI_EXAMPLE_SAMPLES_TO_TRANSFER / 2; t += 1)
    {
 float input = (((float) (freq * t)) * (float) (M_TWOPI)) /
SSI_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       g_src[2 * t]    = (int16_t) ((INT16_MAX * sinf(input)));
       g_src[2 * t + 1] = (int16_t) ((INT16_MAX * sinf(input)));
    }
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Transfer data. */
    (void) R_SSI_WriteRead(&g_i2s_ctrl,
                           (uint8_t *) &g_src[0],
                           (uint8_t *) &g_dest[0],
                           SSI_EXAMPLE_SAMPLES_TO_TRANSFER * sizeof(int16_t));
}
```

### Streaming Example

This is an example of using SSI to stream audio data. This application uses a double buffer to store

PCM sine wave data. It starts transmitting in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart transmission in the main loop.

This example also checks the return code of R_SSI_Write() because R_SSI_Write() can return an error if a transmit overflow occurs before the FIFO is reloaded. If a transmit overflow occurs before the FIFO is reloaded, the SSI will be stopped in the error interrupt, and it cannot be restarted until the I2S_EVENT_IDLE callback is received.

```
#define SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050)
#define SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024)
#define SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t       g_stream_src[2][SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when SSI is ready for more data. */
void ssi_example_callback (i2s_callback_args_t * p_args)
{
 /* Reload the FIFO if we hit the transmit watermark or restart transmission if the
SSI is idle because it was
  * stopped after a transmit FIFO overflow. */
 if ((I2S_EVENT_TX_EMPTY == p_args->event) || (I2S_EVENT_IDLE == p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
      }
    }
}
/* Load the transmit FIFO and check for error conditions. */
void ssi_example_write (void)
```

```
{
 /* Transfer data. This call is non-blocking. */
 fsp_err_t err = R_SSI_Write(&g_i2s_ctrl,
                             (uint8_t *) &g_stream_src[g_buffer_index][0],
                             SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof
(int16_t));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
       g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer only if transmission was successful. */
       g_data_ready = false;
    }
 else
    {
 /* Getting here most likely means a transmit overflow occurred before the FIFO could
be reloaded. The
  * application must wait until the SSI is idle, then restart transmission. In this
example, the idle
  * callback transmits data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void ssi_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK / 2; i += 1)
    {
```

```
 float input = (((float) (freq * t)) * (float) M_TWOPI) /
SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       t++;
 /* Store sample twice, once for left channel and once for right channel. */
       int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
       g_stream_src[buffer_index][2 * i]    = sample;
       g_stream_src[buffer_index][2 * i + 1] = sample;
    }
 /* Data is ready to be sent in the interrupt. */
    g_data_ready = true;
}
void ssi_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       ssi_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
     {
 /* Clear flag. */
           g_send_data_in_main_loop = false;
 /* Reload FIFO and handle errors. */
           ssi_example_write();
     }
 /* If the next buffer is ready, wait for the data to be sent in the interrupt. */
 while (g_data_ready)
     {
```

```
 /* Do nothing. */
      }
    }
}
```

### Data Structures

| | |
|---|---|
| struct | ssi_instance_ctrl_t |
| struct | ssi_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | ssi_audio_clock_t |
| enum | ssi_clock_div_t |

## Data Structure Documentation

### ◆ ssi_instance_ctrl_t

| struct ssi_instance_ctrl_t |
|---|
| Channel instance control block. DO NOT INITIALIZE. Initialization occurs when i2s_api_t::open is called. |

### ◆ ssi_extended_cfg_t

| struct ssi_extended_cfg_t | | |
|---|---|---|
| SSI configuration extension. This extension is optional. | | |
| Data Fields | | |
| ssi_audio_clock_t | audio_clock | Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL. |
| ssi_clock_div_t | bit_clock_div | Select bit clock division ratio. |

## Enumeration Type Documentation

### ◆ ssi_audio_clock_t

| enum ssi_audio_clock_t | |
|---|---|
| Audio clock source. | |
| Enumerator | |
| SSI_AUDIO_CLOCK_EXTERNAL | Audio clock source is the AUDIO_CLK input pin. |
| SSI_AUDIO_CLOCK_INTERNAL | Audio clock source is internal connection to a MCU specific GPT channel output. |

### ◆ ssi_clock_div_t

| enum ssi_clock_div_t | |
|---|---|
| Bit clock division ratio. Bit clock frequency = audio clock frequency / bit clock division ratio. | |
| Enumerator | |
| SSI_CLOCK_DIV_1 | Clock divisor 1. |
| SSI_CLOCK_DIV_2 | Clock divisor 2. |
| SSI_CLOCK_DIV_4 | Clock divisor 4. |
| SSI_CLOCK_DIV_6 | Clock divisor 6. |
| SSI_CLOCK_DIV_8 | Clock divisor 8. |
| SSI_CLOCK_DIV_12 | Clock divisor 12. |
| SSI_CLOCK_DIV_16 | Clock divisor 16. |
| SSI_CLOCK_DIV_24 | Clock divisor 24. |
| SSI_CLOCK_DIV_32 | Clock divisor 32. |
| SSI_CLOCK_DIV_48 | Clock divisor 48. |
| SSI_CLOCK_DIV_64 | Clock divisor 64. |
| SSI_CLOCK_DIV_96 | Clock divisor 96. |
| SSI_CLOCK_DIV_128 | Clock divisor 128. |

## Function Documentation

### ◆ R_SSI_Open()

fsp_err_t R_SSI_Open ( i2s_ctrl_t *const *p_ctrl*, i2s_cfg_t const *const *p_cfg* )

Opens the SSI. Implements i2s_api_t::open.

This function sets this clock divisor and the configurations specified in i2s_cfg_t. It also opens the timer and transfer instances if they are provided.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Ready for I2S communication. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_cfg is null. |
| FSP_ERR_ALREADY_OPEN | The control block has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Channel number is not available on this MCU. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SSI_Write()

fsp_err_t R_SSI_Write ( i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, uint32_t const *bytes* )

Writes data buffer to SSI. Implements i2s_api_t::write.

This function resets the transfer if the transfer interface is used, or writes the length of data that fits in the FIFO then stores the remaining write buffer in the control block to be written in the ISR.

Write() cannot be called if another write(), read() or writeRead() operation is in progress. Write can be called when the SSI is idle, or after the I2S_EVENT_TX_EMPTY event.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write initiated successfully. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_src was null, or bytes requested was 0. |
| FSP_ERR_IN_USE | Another transfer is in progress, data was not written. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_UNDERFLOW | A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_SSI_Read()

fsp_err_t R_SSI_Read ( i2s_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes* )

Reads data into provided buffer. Implements i2s_api_t::read.

This function resets the transfer if the transfer interface is used, or reads the length of data available in the FIFO then stores the remaining read buffer in the control block to be filled in the ISR.

Read() cannot be called if another write(), read() or writeRead() operation is in progress. Read can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read initiated successfully. |
| FSP_ERR_IN_USE | Peripheral is in the wrong mode or not idle. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl or p_dest was null, or bytes requested was 0. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_OVERFLOW | A receive overflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_SSI_WriteRead()

fsp_err_t R_SSI_WriteRead ( i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, void *const *p_dest*, uint32_t const *bytes* )

Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead.

This function calls R_SSI_Write and R_SSI_Read.

writeRead() cannot be called if another write(), read() or writeRead() operation is in progress. writeRead() can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write and read initiated successfully. |
| FSP_ERR_IN_USE | Peripheral is in the wrong mode or not idle. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_UNDERFLOW | A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication. |
| FSP_ERR_OVERFLOW | A receive overflow error is pending. Wait for the SSI to go idle before resuming communication. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reset

### ◆ R_SSI_Stop()

fsp_err_t R_SSI_Stop ( i2s_ctrl_t *const *p_ctrl*)

Stops SSI. Implements i2s_api_t::stop.

This function disables both transmission and reception, and disables any transfer instances used.

The SSI will stop on the next frame boundary. Do not restart SSI until it is idle.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | I2S communication stop request issued. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

**Returns**

See Common Error Codes or lower level drivers for other possible return codes.

### ◆ R_SSI_Mute()

fsp_err_t R_SSI_Mute ( i2s_ctrl_t *const *p_ctrl*, i2s_mute_t const *mute_enable* )

Mutes SSI on the next frame boundary. Implements i2s_api_t::mute.

Data is still written while mute is enabled, but the transmit line outputs zeros.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transmission is muted. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_StatusGet()

fsp_err_t R_SSI_StatusGet ( i2s_ctrl_t *const *p_ctrl*, i2s_status_t *const *p_status* )

Gets SSI status and stores it in provided pointer p_status. Implements i2s_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored successfully. |
| FSP_ERR_ASSERTION | The p_instance_ctrl or p_status parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_Close()

fsp_err_t R_SSI_Close ( i2s_ctrl_t *const *p_ctrl*)

Closes SSI. Implements i2s_api_t::close.

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device closed successfully. |
| FSP_ERR_ASSERTION | The pointer to p_ctrl was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

### ◆ R_SSI_CallbackSet()

| fsp_err_t R_SSI_CallbackSet ( i2s_ctrl_t *const *p_api_ctrl*, void(*)(i2s_callback_args_t *) *p_callback*, void *const *p_context*, i2s_callback_args_t *const *p_callback_memory* ) |
|---|
| Updates the user callback and has option of providing memory for callback structure. Implements i2s_api_t::callbackSet |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.6.16 I3C (r_i3c)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_I3C_Open (i3c_ctrl_t *const p_api_ctrl, i3c_cfg_t const *const p_cfg) |
| fsp_err_t | R_I3C_Enable (i3c_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_I3C_DeviceCfgSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_cfg_t const *const p_device_cfg) |
| fsp_err_t | R_I3C_MasterDeviceTableSet (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg) |
| fsp_err_t | R_I3C_SlaveStatusSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_status_t status) |
| fsp_err_t | R_I3C_DeviceSelect (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, uint32_t bitrate_mode) |
| fsp_err_t | R_I3C_DynamicAddressAssignmentStart (i3c_ctrl_t *const p_api_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count) |
| fsp_err_t | R_I3C_CommandSend (i3c_ctrl_t *const p_api_ctrl, i3c_command_descriptor_t const *const p_command_descriptor) |

| | |
|---|---|
| fsp_err_t | R_I3C_Write (i3c_ctrl_t *const p_api_ctrl, uint8_t const *const p_data, uint32_t length, bool restart) |
| fsp_err_t | R_I3C_Read (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t length, bool restart) |
| fsp_err_t | R_I3C_IbiWrite (i3c_ctrl_t *const p_api_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length) |
| fsp_err_t | R_I3C_IbiRead (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t length) |
| fsp_err_t | R_I3C_Close (i3c_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the I3C peripheral on RA MCUs. This module implements the I3C Interface.

# Overview

I3C is a communication protocol defined by MIPI that aims to improve on I2C by increasing the maximum transfer rate, as well as providing other features like "In-band Interrupts", "Dynamic Address Assignment", and a set of standard "Common Command Codes".

### Features

- I3C Master Mode
- I3C Slave Mode
- Dynamic Address Assignment (ENTDAA/SETDASA)
- SDR Read/Write transfers
- HDR-DDR Read/Write Commands
- I2C Legacy Read/Write transfers
- In-Band Interrupts (Interrupt Requests, Hot-Join Requests)
- Common Command Codes
- Clock Stalling
- Timeout Detection

# Master Mode

On an I3C bus, only one device may operate in master mode at a time. The current master is responsible for initiating I2C Legacy transfers, SDR transfers, Common Command Codes, and handling IBIs (Interrupt Requests, Hot-Join Requests). In order to perform these operations, the driver has an internal device table that is used for storing configuration information for each device on the bus (See i3c_device_table_cfg_t). Each entry in the device table contains the static or dynamic address of the device, and IBI permissions for accepting or rejecting IBI requests from the device. The device table has a limited number of entries as well as one extended device entry that only contains the static or dynamic address of a device (See below).

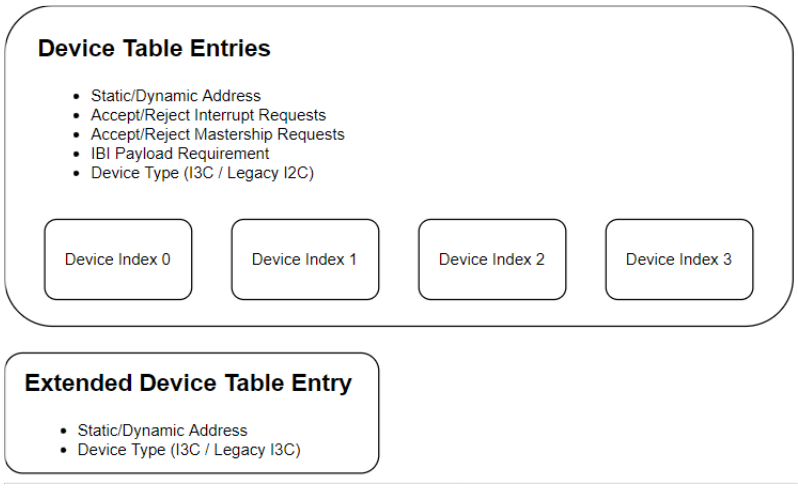



Figure 222: Master Device Table

In order to initiate I2C Legacy transfers, SDR transfers, or Common Command Codes, the master must select a device entry from the device table using i3c_api_t::deviceSelect. Once a device has been selected, all subsequent operations will be directed to the selected device until a new device is selected using i3c_api_t::deviceSelect.

The master may also receive IBI requests that are initiated by slave devices on the bus. If there is a payload, then the driver will write the data into a buffer that is provided by the application by calling i3c_api_t::ibiRead. If the application has not provided an IBI buffer prior to receiving an IBI, then the it will get a callback requesting an IBI buffer. Once the IBI is completed, the application will be notified by a callback.

*Note*

*1. Even though there are only a limited number of device table entries and one extended device table entry, the application can operate on more devices by maintaining its own list of devices and updating the extended device entry as needed. Note however that devices defined in the extended device table entry will not be able to initiate IBI requests.*

### Main Master

The main master is responsible for configuring the dynamic address of all devices on the bus. The driver initiates this procedure by calling i3c_api_t::dynamicAddressAssignmentStart. Before starting address assignment, the application must configure the device table using i3c_api_t::masterDeviceTableSet.

### Enter Dynamic Address Assignment (ENTDAA):

The application initiates the ENTDAA operation by calling i3c_api_t::dynamicAddressAssignmentStart with a starting index into the master device table and a count specifying the number of devices to configure. The master starts by sending the ENTDAA command. Every I3C device on the bus that has not already been initialized will acknowledge the command and attempt to write its Provisional ID, DCR, and BCR registers. The device with the smallest value in these registers will win arbitration and be assigned with the first dynamic address defined in the master device table. The master will then increment the index and repeat the process by assigning the dynamic address to the next device. The process continues until the specified number of devices have been initialized or until there are no more devices to configure.

*Note*

*1. The IBI payload setting will automatically be updated in the master device table based on the BCR setting that was read during ENTDAA.*
*2. After each device successfully writes its Provisional ID, DCR, and BCR registers, the application will get a callback that will provide the value of the registers.*
*3. If the starting index is set to the extended device entry, then the device count must be set to 1.*
*4. The main master assigns its own dynamic address with i3c_api_t::deviceCfgSet.*

**Set Dynamic Address from Static Address (SETDASA):**

The application initiates the SETDASA operation by calling i3c_api_t::dynamicAddressAssignmentStart with an index into the master device table. The master sends the SETDASA command to the static address defined in the given device table entry, and then assigns the associated dynamic address.

*Note*

*1. Set the count to 0 when using SETDASA.*

# Slave Mode

In slave mode, the device configures its static address, Provisional ID, BCR, and DCR registers using i3c_api_t::deviceCfgSet, and then waits for the master to initiate communication. Prior to being assigned a dynamic address, the slave will operate as an I2C device using its static address. The application will receive a callback when the master assigns it a dynamic address, after which point, the slave will operate as an I3C device until it receives the RSTDAA command.

Depending on the capabilities defined in its BCR register, the slave may also initiate IBI Interrupt Requests, and Hot-Join Requests using i3c_api_t::ibiWrite

# Configuration

**Build Time Configurations for r_i3c**

The following build time configurations are defined in fsp_cfg/r_i3c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Unaligned Buffer Support | • Enabled<br>• Disabled | Enabled | Unaligned buffer support may be optionally disabled for improved performance. |
| Master Support | • Enabled<br>• Disabled | Enabled | If only slave mode is required, disable master support to decrease code size. |
| Slave Support | • Enabled<br>• Disabled | Enabled | If only master mode is required, disable slave |

| | | | |
|---|---|---|---|
| | | | support to decrease code size. |
| Error Recovery Procedure | MCU Specific Options | | There are two different chip versions of the RA2E2 MCU (Version 1 and Version 2). Each version requires a different procedure for recovering from errors. |

### Configurations for Connectivity > I3C (r_i3c)

This module can be added to the Stacks tab via New Stack > Connectivity > I3C (r_i3c).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Bitrate Settings | | | |
| Bitrate Settings > Standard Mode | | | |
| Bitrate Settings > Standard Mode > Open-Drain | | | |
| Logic High Period (ns) | Must be an integer greater than 0. | 167 | The Logic High period of SCL during Standard Mode Open Drain transfers. |
| Frequency | Must be an integer greater than 0. | 1000000 | The Frequency of SCL during Standard Mode Open Drain transfers. |
| Bitrate Settings > Standard Mode > Push-Pull | | | |
| Logic High Period (ns) | The Logic High Period must be greater than or equal to 24 Nanoseconds. | 167 | The Logic High period of SCL during Standard Mode Push Pull transfers. |
| Frequency | Push-Pull frequency must be greater than or equal to 10000 Hz. | 3400000 | The Frequency of SCL during Standard Mode Push-Pull transfers. |
| Bitrate Settings > Extended Mode | | | |
| Bitrate Settings > Extended Mode > Open-Drain | | | |
| Logic High Period (ns) | Must be an integer greater than 0. | 167 | The Logic High period of SCL during Extended Mode Open Drain transfers. |
| Frequency | Must be an integer greater than 0. | 1000000 | The Frequency of SCL during Extended Mode Open Drain transfers. |
| Bitrate Settings > Extended Mode > Push-Pull | | | |
| Logic High Period (ns) | The Logic High Period must be greater than | 167 | The Logic High period of SCL during Extended |

| | | | |
|---|---|---|---|
| | or equal to 24 Nanoseconds. | | Mode Push Pull transfers. |
| Frequency | Push-Pull frequency must be greater than or equal to 10000 Hz. | 3400000 | The Frequency of SCL during Extended Mode Push-Pull transfers. |
| Bitrate Settings > Bus Timing | | | |
| Open Drain Rising Time (ns) | Rising time must be greater than or equal to 0 nanoseconds. | 0 | The Open Drain rising time in nanoseconds. |
| Open Drain Falling Time (ns) | Falling time must be greater than or equal to 0 nanoseconds. | 0 | The Open Drain falling time in nanoseconds. |
| Push-Pull Rising Time (ns) | Rising time must be greater than or equal to 0 nanoseconds. | 0 | The Push-Pull rising time in nanoseconds. |
| Push-Pull Falling Time (ns) | Falling time must be greater than or equal to 0 nanoseconds. | 0 | The Push-Pull rising time in nanoseconds. |
| Bitrate Settings > Clock Stalling | | | |
| Address Assignment Phase | • Enabled<br>• Disabled | Disabled | Enable clock stalling during the Address Assignment Phase of ENTDAA. |
| Transition Phase | MCU Specific Options | | Enable clock stalling during the Transition Bit of a read transfer. |
| Parity Phase | • Enabled<br>• Disabled | Disabled | Enable clock stalling during the Parity Bit of a write transfer. |
| Ack Phase | • Enabled<br>• Disabled | Disabled | Enable clock stalling during the ACK phase of a transfer. |
| Time (us) | Must be greater than or equal to 0. | 0 | The amount of time to stall the clock during the Address Assignment Phase, Transition Phase, Parity Phase, and ACK Phase. |
| Master Mode | | | |
| ACK Hot-Join Requests | • Enabled<br>• Disabled | Disabled | If enabled, the I3C instance will ACK Hot-Join Requests and notify the application. |
| Notify Rejected Hot-Join Requests. | • Enabled<br>• Disabled | Disabled | If enabled, the application will get a |

| | | | |
|---|---|---|---|
| | | | callback when an IBI Hot-Join Request is rejected. |
| Notify Rejected Mastership Requests. | • Enabled<br>• Disabled | Disabled | If enabled, the application will get a callback when an IBI Mastership Request is rejected. |
| Notify Rejected Interrupt Requests. | • Enabled<br>• Disabled | Disabled | If enabled, the application will get a callback when an IBI Interrupt Request is rejected. |
| Slave Mode | | | |
| Slave Mode > Command Response Info | | | |
| Slave Mode > Command Response Info > ENEC/DISEC | | | |
| In-Band Interrupts | • Enabled<br>• Disabled | Disabled | Configure whether the slave can issue IBI requests. |
| Hot-Join Requests | • Enabled<br>• Disabled | Disabled | Configure whether the slave can issue Hot-Join requests. |
| Slave Mode > Command Response Info > ENTASn | | | |
| Activity State | • Activity State 0<br>• Activity State 1<br>• Activity State 2<br>• Activity State 3 | Activity State 0 | Configure the starting activity state of the slave. |
| Slave Mode > Command Response Info > SETMWL/GETMWL | | | |
| Max Write Length | Write length must be in the range of [8, 65535]. | 65535 | Set the max write length. |
| Slave Mode > Command Response Info > SETMRL/GETMRL | | | |
| Max Read Length | Read length must be in the range of [16, 65535]. | 65535 | Set the max read length. |
| Max IBI Payload Length | Read length must be in the range of [0, 255]. | 0 | Set the max IBI payload length, or set it to 0 for unlimited. |
| Slave Mode > Command Response Info > GETMXDS | | | |
| Write Data Rate | • FSCL_MAX<br>• 8Mhz<br>• 6Mhz<br>• 4Mhz<br>• 2Mhz | 2Mhz | Set the max write data rate. |

| | | | |
|---|---|---|---|
| Read Data Rate | • FSCL_MAX<br>• 8Mhz<br>• 6Mhz<br>• 4Mhz<br>• 2Mhz | 2Mhz | Set the max read data rate. |
| Clock to Data Turnaround Time | • 8 Nanoseconds<br>• 9 Nanoseconds<br>• 10 Nanoseconds<br>• 11 Nanoseconds<br>• 12 Nanoseconds<br>• Greater than 12 Nanoseconds | 8 Nanoseconds | Set the clock to data turnaround time. |
| Include Max Read Turnaround Time | • Enabled<br>• Disabled | Disabled | Configure whether the Max Read Turnaround time will be transmitted. |
| Max Read Turnaround Time | Value must be in the range [0, 255]. | 0 | Set max read turnaround time. |
| Slave Mode > Command Response Info > GETXTIME | | | |
| Frequency Byte | Value must be in the range [0, 255]. | 0 | Set the internal oscillator frequency in increments of 0.5 Mhz. |
| Inaccuracy Byte | Value must be in the range [0, 255]. | 0 | Set the oscillator inaccuracy byte in increments of 0.5% |
| Slave Mode > Command Response Info > GETCAP/GETHDRCAP | | | |
| HDR-DDR (Mode 0) | MCU Specific Options | | Slave supports HDR-DDR (Mode 0) |
| HDR-TSP (Mode 1) | MCU Specific Options | | Slave supports HDR-TSP (Mode 1) |
| HDR-TSL (Mode 2) | MCU Specific Options | | Slave supports HDR-TSL (Mode 2) |
| Interrupts | | | |
| Interrupt Priority | MCU Specific Options | | The interrupt priority of the RX, TX, RESPONSE, RCV_STATUS, and IBI ISRs. |
| Error and Event Interrupt Priority | MCU Specific Options | | The interrupt priority of the EEI ISR which is used to notify the application when an Internal Error, HDR Exit Pattern, or Timeout is detected. |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_i3c0 | Module name. |
| Callback | Name must be a valid C symbol | g_i3c0_callback | A user callback function must be provided. This will be called in order to notify the application of I3C events and provide status information. |
| Device Type | • Main Master<br>• Slave | Slave | The role that the I3C instance will take on the I3C bus. |
| Bus Free Condition Detection Time (ns) | Must be greater than or equal to 38.4 nanoseconds. | 38.4 | The minimum period occurring after a STOP and before a START. |
| Bus Available Condition Detection Time (us) | Must be greater than or equal to 1 microsecond. | 1 | The minimum period occurring after the Bus Free Condition when Slaves can initiate IBI requests. |
| Bus Idle Condition Detection Time (us) | Must be greater than or equal to 1000 microseconds. | 1000 | The minimum period occurring after the Bus Available Condition when Slaves can initiate Hot-Join requests. |
| Timeout Detection | • Enabled<br>• Disabled | Disabled | If enabled, the application will get a callback if SCL is stuck at a logic high or logic low level for more than 65535 cycles of the I3C source clock. |

### Clock Configuration

The following settings are used to configure the timing of SCL.

- Frequency of TCLK
- Standard Mode
  - Open Drain High Period ($T_{HIGH}$)
  - Open Drain Frequency
  - Push-Pull High Period ($T_{HIGH}$)
  - Push-Pull Frequency
- Extended Mode
  - Open Drain High Period ($T_{HIGH}$)
  - Open Drain Frequency
  - Push-Pull High Period ($T_{HIGH}$)
  - Push-Pull Frequency
- i3c_bitrate_mode_t (Set during i3c_api_t::deviceSelect)

The Standard and Extended Mode settings define two separate SCL configurations that can be selected at run-time using i3c_api_t::deviceSelect.

In addition to selecting between the Standard and Extended Mode settings, the base SCL period can also be multiplied using the following options:

- I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2: Multiple the base Standard Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2: Multiple the base Extended Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4: Multiple the base Extended Open Drain and Push-Pull period by 4.

In order to get accurate frequency calculations, the Rising and Falling edges must be input into the calculation. These values will depend on the topology the I3C bus that will be different for every application.

*Note*

*1. The Standard and Extended Open Drain period settings define the period to use during legacy I2C transfers (Only use the following i3c_bitrate_mode_t settings with I2C transfers: I3C_BITRATE_MODE_I2C_STDBR, I3C_BITRATE_MODE_I2C_EXTBR).*
*2. $T_{HIGH}$ is defined in Figure 31 in the MIPI I3C Specification v1.0 and describes the Logic High period.*
*3. Depending on the MCU, TCLK is either derived from PCLKD or from a dedicated I3C Clock (I3CCLK).*

### Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I3C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

## Usage Notes

### Read and Write in Slave Mode

In slave mode, calling read or write does not start a transfer. Instead, calling read or write will configure the driver to perform the next read or write transfer using the user provided buffer.

Provided that a transfer is not already in progress, i3c_api_t::read and i3c_api_t::write can be called to update the internal buffers even if the transfer has not been completed yet. Both the read and write buffers can be configured at the same time in order to prepare the driver for when the master initiates a read or a write transfer.

If there is no space remaining in a user configured read buffer, the application will get a I3C_EVENT_READ_BUFFER_FULL callback requesting for a new read buffer to be provided.

### Event Status

When a write, read, ibiWrite, ibiRead or commandSend, operation is completed, the i3c_callback_args_t::event_status should be checked. The event_status will provide information about the success or failure of the operation.

The following are possible statuses:

- I3C_EVENT_STATUS_SUCCESS

- I3C_EVENT_STATUS_PARITY
- I3C_EVENT_STATUS_FRAME
- I3C_EVENT_STATUS_ADDRESS_HEADER
- I3C_EVENT_STATUS_NACK
- I3C_EVENT_STATUS_OVERFLOW
- I3C_EVENT_STATUS_ABORTED_TO_MASTER
- I3C_EVENT_STATUS_ABORTED

*Note*

*In master mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will be I3C_EVENT_STATUS_ABORTED.*
*In slave mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will be I3C_EVENT_STATUS_ABORTED_TO_MASTER.*

### Direct Get Common Command Codes in Slave Mode

When an I3C slave receives a Command Code of type Direct Get, the response is automatically sent from the device Special Function Registers (SFR). The SFR contains information for each command code and can be configured during open (See i3c_extended_cfg_t::slave_command_response_info). This allows the slave to respond to Direct Get Command Codes much faster, and removes the burden of responding to these commands from the application.

The response to the GETSTATUS command can be configured at run-time using i3c_api_t::slaveStatusSet.

### Disabling Unaligned Buffer Support

Support for performing read and write operations on unaligned buffers can be disabled in order to improve performance. When unaligned buffer support is disabled, all buffers passed to read, ibiRead, write, ibiWrite, and commandSend must be aligned to 4 bytes and the size of the buffers must be a multiple of 4 bytes.

In master mode, the value of the length passed to i3c_api_t::read and i3c_api_t::write sets the total length of the operation in bytes. During the read or write operation, the driver may read or write to the last word of memory during the operation. This means that the allocated memory for the buffer passed to read and write needs to be a multiple of 4 bytes even though the transfer length is not a multiple of 4 bytes.

In slave mode, the length passed to i3c_api_t::read must be a multiple of 4 bytes. The length passed to i3c_api_t::write sets the number of bytes that the slave will write. The size of the buffer passed to write still needs to be a multiple of 4 bytes.

### Max Data Speed Limitation on RA2E2/RA2L2 MCUs

In Slave Mode, it is highly recommended that BCR[0] be set to '1' in order to indicate to the master that the device doesn't support the max data speed. The master is then required to use the GETMXDS command to get the device specific data speed limitations.

This will allow the slave to specify its maximum supported data rate for read/write, and the maximum read turnaround time (See GETMXDS in the MIPI I3C Specification v1.0).

### Mixed Fast Bus

The MIPI I3C Specification v1.0 defines a Mixed Fast Bus as a bus that has legacy I2C devices that all have a 50ns Spike Filter.

On Mixed Fast Buses, SCL has the following constraints during I3C SDR transfers:

- SCL High Period: $t_{DIG_H_MIXED(MIN)}$ to $t_{DIG_H_MIXED(MAX)}$
- SCL Low Period: Up to $t_{DIG_L(MAX)}$

In this case, configure the Extended Bitrate settings for I3C SDR transfers, and use the Standard Bitrate settings for I2C transfers.

*Note*

*1. See section 5.1.2.4 in the MIPI I3C Specification v1.0.*

### Mixed Slow Bus

The MIPI I3C Specification v1.0 defines a Mixed Slow Bus as a bus that has legacy I2C devices that do not have a 50ns Spike Filter.

In this case, the SCL frequency is limited to I2C Fast Mode or I2C Fast Mode Plus.

*Note*

*1. See section 5.1.2.4 in the MIPI I3C Specification v1.0.*

### RA2E2 MCU Version

There are two versions of the RA2E2. The version number can be identified by the part number or by reading the MCU Version Register.

| Part Number | Chip Version |
|---|---|
| R7FA2E2A72DNK::HA0 | Version 1 |
| R7FA2E2A72DNK::HA1 | Version 2 |
| R7FA2E2A72DNK::AA0 | Version 1 |
| R7FA2E2A72DNK::AA1 | Version 2 |

The I3C peripheral on the RA2E2 Version 1 has a hardware limitation related to using dynamic address assignment in slave mode. When the ENTDAA command is received, the peripheral will continue driving its PID, DCR, and BCR registers during arbitration even after its dynamic address has already been configured. This will prevent any other devices on the bus from being configured after the RA2E2 Version 1 device has been configured. One workaround for this limitation is to configure the Provisional ID to all '1's in order to ensure that the RA2E2 Version 1 device is configured last. If more than one RA2E2 Version 1 device is present on the same bus, consider initializing them using another method (Eg. SETDASA). **Note that this issue has been corrected on the RA2E2 Version 2 (Part numbers ending in #AA1 or #HA1).**

In addition to fixing this hardware limitation, the revised I3C peripheral on the RA2E2 version 2 also greatly simplifies the error recovery procedures. By default, the driver will automatically include the error recovery procedures for both versions, however, it can be configured to only support version 1 or version 2 procedures in order to reduce code size.

### Limitations

Developers should be aware of the following limitations when using the I3C:

- The MIPI Reserved area and Vendor Extension area of Command Codes are not supported.

- Mixed Fast Bus topology has the following limitation on RA2E2/RA2L2 MCUs. The minimum SCL high period ($T_{DIG_H}$) is 156 nanoseconds when PCLKD is 48 Mhz, and 120 nanoseconds when PCLKD is 64 Mhz. On a Mixed Fast Bus, the high period $T_{DIG_HIGH_MIXED}$ must be less than 45 nanoseconds in order to ensure that Legacy I2C devices do not interpret I3C signaling as valid I2C signaling (See Table 111 Push-Pull-Timing Parameters in the MIPI I3C Specification v1.1). This required high period cannot be achievable with RA2E2/RA2L2 MCUs.
- Secondary Master device role is not currently supported.

# Examples

### I3C Master Basic Example

This is a basic example of minimal use of the I3C Master in an application.

```
void i3c_master_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = EXAMPLE_DYNAMIC_ADDRESS,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I3C,
```

```
      .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
using ENTDAA.
  */
      .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
      .master_request_accept = false,
   };
 /* Set the device configuration in the master device table. */
   status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
   assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
   status = R_I3C_Enable(&g_i3c_ctrl);
   assert(FSP_SUCCESS == status);
 /* Start assigning dynamic addresses to devices on the bus using the ENTDAA command.
*/
   status = R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl,
I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA, 0, 1);
   assert(FSP_SUCCESS == status);
 /* Wait for dynamic address assignment to complete. */
   i3c_app_event_wait(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 /* Select the configured device and bitrate mode for the following operations. */
   status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I3C_SDR0_STDBR);
   assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_buffer[] = {1, 2, 3, 4, 5};
   status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
   assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
   i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_buffer[16];
   status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
```

```
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
/* This function is called by the I3C driver from ISRs in order to notify the
application of I3C events. */
void i3c_master_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_ENTDAA_ADDRESS_PHASE:
      {
 /* The device PID, DCR, and BCR registers will be available in
i3c_callback_args_t::p_slave_info. */
 break;
       }
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
       }
 case I3C_EVENT_WRITE_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
       }
 default:
```

```
     {
 break;
      }
   }
}
```

### I3C Slave Basic Example

This is a basic example of minimal use of the I3C Slave in an application.

```
void i3c_slave_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t slave_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_STATIC_ADDRESS,
 /* The dynamic address will be automatically updated when the master configures this
device using ENTDAA. */
       .dynamic_address = 0,
 /* Device Registers that are read by the master. */
       .slave_info      =
     {
           .bcr = EXAMPLE_BCR_SETTING,
           .dcr = EXAMPLE_DCR_SETTING,
           .pid =
     {
               0, 1, 2, 3, 4, 5
        }
      }
   };
 /* Set the device configuration for this device. */
```

```
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &slave_device_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable Slave Mode. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 static uint8_t p_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 static uint8_t p_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
 /* Set the buffer for storing data received during a read transfer. */
    status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a read transfer. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
 /* Set the write buffer that will be transmitted during a write transfer. */
    status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a write transfer. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
}
void i3c_slave_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
     {
           i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
      }
 case I3C_EVENT_READ_BUFFER_FULL:
     {
 /* If there is no user provided read buffer, or if the user provided read buffer has
been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::read. If no read buffer is provided, then
```

```
any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_Read(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE, false);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read by the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 case I3C_EVENT_WRITE_COMPLETE:
      {
 /* The number of bytes written by the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 default:
      {
 break;
       }
    }
}
```

### I2C Legacy Basic Example

This is a basic example of minimal use of I2C Legacy transfers in an application.

```
void i2c_legacy_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = 0,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I2C,
 /* These options are not used in I2C. */
       .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
using ENTDAA.
  */
       .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
       .master_request_accept = false,
```

```
    };
 /* Set the device configuration in the master device table. */
    status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 /* Select the configured device for the following operations. */
    status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I2C_STDBR);
    assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_data[] = {1, 2, 3, 4, 5};
    status = R_I3C_Write(&g_i3c_ctrl, p_write_data, sizeof(p_write_data), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_data[16];
    status = R_I3C_Read(&g_i3c_ctrl, p_read_data, sizeof(p_read_data), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
void i2c_legacy_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_WRITE_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
```

```
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

### I3C Master In-band Interrupts Example

This is a basic example of reading In-band Interrupts in I3C Master mode.

```
void i3c_master_ibi_basic_example (void)
{
 static uint8_t p_ibi_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Set the buffer for storing IBI data that is read from the slave. */
 fsp_err_t status = R_I3C_IbiRead(&g_i3c_ctrl, p_ibi_read_buffer, sizeof
(p_ibi_read_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiRead transfer to complete.
  * Note that the master does not need to wait for the IBI, and can start other
operations. */
    i3c_app_event_wait(I3C_EVENT_IBI_READ_COMPLETE);
}
void i3c_master_ibi_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_IBI_READ_BUFFER_FULL:
      {
 /* If there is no user provided ibiRead buffer, or if the user provided ibiRead
```

```
buffer has been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::ibiRead. If no read buffer is provided,
then any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_IbiRead(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE);
 break;
       }
 case I3C_EVENT_IBI_READ_COMPLETE:
      {
 /* When an IBI is completed, the transfer_size, ibi_type, and ibi_address will be
available in p_args. */
 switch (p_args->ibi_type)
      {
 case I3C_IBI_TYPE_INTERRUPT:
      {
 /* Notify the application that an IBI was read. */
                   i3c_app_event_notify(I3C_EVENT_IBI_READ_COMPLETE);
 break;
         }
 case I3C_IBI_TYPE_HOT_JOIN:
      {
 /* If a Hot-Join event is received, then the master can initiate the dynamic address
assignment procedure. */
 R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl, I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA,
0, 1);
 break;
         }
 default:
      {
 break;
         }
```

```
        }
      }
 default:
     {
 break;
      }
   }
}
```

### I3C Slave In-band Interrupts Example

This is a basic example of writing In-band Interrupts in I3C Slave mode.

```
void i3c_slave_ibi_write_basic_example (void)
{
    uint8_t ibi_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE] = {0};
 /* Initiate an In-band interrupt in slave mode.
  * Note: If the slave does not have an IBI payload or if it is a Hot-Join request,
the write buffer should be set
  * to NULL and the write length should be set to 0. */
 fsp_err_t status = R_I3C_IbiWrite(&g_i3c_ctrl, I3C_IBI_TYPE_INTERRUPT,
ibi_write_buffer, sizeof(ibi_write_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiWrite transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_IBI_WRITE_COMPLETE);
}
void i3c_slave_ibi_write_basic_example_callback (i3c_callback_args_t const * const
p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_IBI_WRITE_COMPLETE:
     {
 /* Notify the application that the IBI write is complete. */
           i3c_app_event_notify(I3C_EVENT_IBI_WRITE_COMPLETE);
```

```
 break;
       }
 default:
      {
 break;
       }
    }
}
```

### I3C Master Common Command Codes Example

This is a basic example of sending Common Command Codes in I3C Master mode.

```
void i3c_master_ccc_example (void)
{
 static uint8_t command_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Setup the command descriptor. */
 static i3c_command_descriptor_t command_descriptor =
    {
       .command_code = I3C_CCC_DIRECT_GETSTATUS,
 /* Set a buffer for storing the data read by the command. */
       .p_buffer    = command_buffer,
 /* The length for a GETSTATUS command is 2 bytes. */
       .length       = 2,
 /* Terminate the transfer with a STOP condition. */
       .restart      = false,
 /* The GETSTATUS command is a Direct Get Command so rnw should be true. */
       .rnw          = true,
    };
 /* Send the command. */
 fsp_err_t status = R_I3C_CommandSend(&g_i3c_ctrl, &command_descriptor);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
 /* The command_buffer will have the status info that was read from the slave device.
```

```
*/
}
void i3c_master_ccc_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* Notify the application that the command is complete. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

### I3C Slave Common Command Codes Example

This is a basic example of receiving Common Command Codes in I3C Slave mode.

```
void i3c_slave_ccc_example (void)
{
 static uint8_t read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Broadcast and Direct Set commands will be read into the read_buffer the same way
that
  * a normal SDR Master Write / Slave Read transfer is read. */
 fsp_err_t status = R_I3C_Read(&g_i3c_ctrl, read_buffer, sizeof(read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
}
void i3c_slave_ccc_example_callback (i3c_callback_args_t const * const p_args)
```

```
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* The command code and transfer size will be available in p_args.
  * If the command code is a Broadcast or Direct Set, then data will
  * be stored in the read buffer provided by i3c_api_t::read.
  * If the command code is a Direct Get, then the data will be automatically
  * sent from device SFR. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

### I3C HDR Command Example

This is a basic example of sending an HDR command

```
#define I3C_HDR_COMMAND_CODE (0x55)
#define I3C_HDR_COMMAND_LENGTH (16)
void i3c_hdr_command_example (void)
{
 static uint8_t command_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Setup the command descriptor. */
 static i3c_command_descriptor_t command_descriptor =
    {
 /* Specify the command code for the HDR transfer. */
       .command_code = I3C_HDR_COMMAND_CODE,
 /* Set a buffer for writing command data. */
```

```
      .p_buffer    = command_buffer,
 /* The length for command (HDR commands must be a multiple of 2 bytes). */
      .length       = I3C_HDR_COMMAND_LENGTH,
 /* Terminate the transfer with a HDR-EXIT. */
      .restart      = false,
 /* Specify if this is a read/write command. */
      .rnw          = false,
   };
 /* Select the HDR mode for subsequent transfers. */
 fsp_err_t status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0,
I3C_BITRATE_MODE_I3C_HDR_DDR_STDBR);
   assert(FSP_SUCCESS == status);
 /* Send the command. */
   status = R_I3C_CommandSend(&g_i3c_ctrl, &command_descriptor);
   assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
   i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
}
void i3c_hdr_command_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
   {
 case I3C_EVENT_COMMAND_COMPLETE:
     {
 /* Notify the application that the command is complete. */
           i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
      }
 default:
     {
 break;
      }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | i3c_clock_stalling_t |
| struct | i3c_bitrate_settings_t |
| struct | i3c_ibi_control_t |
| struct | i3c_slave_command_response_info_t |
| struct | i3c_instance_ctrl_t |
| struct | i3c_extended_cfg_t |

## Macros

| | |
|---|---|
| #define | I3C_ERROR_RECOVERY_VERSION_1<br>Support error recovery procedure for chip version 1. More... |
| #define | I3C_ERROR_RECOVERY_VERSION_2<br>Support error recovery procedure for chip version 2. |
| #define | I3C_ERROR_RECOVERY_VERSION_BOTH<br>Support error recovery procedure for chip version 1 and version 2. |
| #define | I3C_DEVICE_INDEX_EXTENDED_DEVICE |
| #define | I3C_EVENT_STATUS_SUCCESS<br>The transfer was completed as expected. More... |
| #define | I3C_EVENT_STATUS_PARITY<br>A parity error was detected. |
| #define | I3C_EVENT_STATUS_FRAME<br>A frame error was detected. |
| #define | I3C_EVENT_STATUS_ADDRESS_HEADER<br>An Address Header error wasdetected. |

| | |
|---|---|
| #define | I3C_EVENT_STATUS_NACK<br>The transfer was NACK'd. |
| #define | I3C_EVENT_STATUS_OVERFLOW<br>A Receive FIFO overflow or Transmit FIFO underflow occurred. |
| #define | I3C_EVENT_STATUS_ABORTED_TO_MASTER<br>In slave mode, the write transfer was ended via the 'T' bit. |
| #define | I3C_EVENT_STATUS_ABORTED<br>In master mode, the transfer was aborted. |
| #define | I3C_EVENT_STATUS_NOT_SUPPORTED<br>Operation is not supported. |
| #define | I3C_EVENT_STATUS_IBI_NACK_DISABLED<br>An IBI was NACK'd and the a DISEC command was sent. |

### Enumerations

| | |
|---|---|
| enum | i3c_bitrate_mode_t |
| enum | i3c_activity_state_t |
| enum | i3c_data_rate_setting_t |
| enum | i3c_clock_data_turnaround_t |

## Data Structure Documentation

### ◆ i3c_clock_stalling_t

| struct i3c_clock_stalling_t | | |
|---|---|---|
| Clock stalling settings. | | |
| Data Fields | | |
| uint32_t | assigned_address_phase_enable: 1 | Enable Clock Stalling during the address phase of the ENTDAA command. |
| uint32_t | transition_phase_enable: 1 | Enable Clock Stalling during the transition bit in read transfers. |

| | | |
|---|---|---|
| uint32_t | parity_phase_enable: 1 | Enable Clock Stalling during the parity bit period in write transfers. |
| uint32_t | ack_phase_enable: 1 | Enable Clock Stalling during the ACK/NACK phase. |
| uint16_t | clock_stalling_time | The amount of time to stall the clock in I3C source clock ticks. |

### ◆ i3c_bitrate_settings_t

| struct i3c_bitrate_settings_t | | |
|---|---|---|
| Bitrate settings for configuring the SCL clock frequency. | | |
| Data Fields | | |
| uint32_t | stdbr | The standard bitrate settings. |
| uint32_t | extbr | The extended bitrate settings. |
| i3c_clock_stalling_t | clock_stalling | Clock Stalling settings (See Master Clock Stalling in the MIPI I3C Specification v1.0). |

### ◆ i3c_ibi_control_t

| struct i3c_ibi_control_t | | |
|---|---|---|
| Settings for controlling the drivers behavior in response to IBIs. | | |
| Data Fields | | |
| uint32_t | hot_join_acknowledge: 1 | If false, NACK all Hot Join requests. |
| uint32_t | notify_rejected_hot_join_requests: 1 | Notify the application when an IBI Hot-Join request has been NACK'd. |
| uint32_t | notify_rejected_mastership_requests: 1 | Notify the application when an IBI Mastership request has been NACK'd. |
| uint32_t | notify_rejected_interrupt_requests: 1 | Notify the application when an IBI Interrupt request has been NACK'd. |

### ◆ i3c_slave_command_response_info_t

| struct i3c_slave_command_response_info_t | | |
|---|---|---|
| Default configuration settings for the slave response to Direct Get Common Command Codes. | | |
| Data Fields | | |
| bool | inband_interrupt_enable | Enable IBI interrupts.<br><br>Slave Event Settings (See ENEC and DISEC in the MIPI I3C |

| | | |
|---|---|---|
| | | Specification v1.0). |
| bool | mastership_request_enable | Enable Mastership requests. |
| bool | hotjoin_request_enable | Enable Hot-Join requests. |
| i3c_activity_state_t | activity_state | Starting Activity State (See ENTASn in the MIPI I3C Specification v1.0). |
| uint16_t | write_length | Max Write Length (See SETMWL and GETMWL in the MIPI I3C Specification v1.0). |
| uint16_t | read_length | Max Read Length (See SETMRL and GETMRL in the MIPI I3C Specification v1.0). |
| uint8_t | ibi_payload_length | Number of bytes that will be written by an IBI (See SETMRL and GETMRL in the MIPI I3C Specification v1.0). |
| i3c_data_rate_setting_t | write_data_rate | Max Write Data Rate.<br><br>Max Data Rate Settings (See GETMXDS in the MIPI I3C Specification v1.0). |
| i3c_data_rate_setting_t | read_data_rate | Max Read Data Rate. |
| i3c_clock_data_turnaround_t | clock_data_turnaround | Max Data Speed Turnaround. |
| bool | read_turnaround_time_enable | Enable transmission of the of the Max Read Max Read Turnaround Time. |
| uint32_t | read_turnaround_time | Max Read Turnaround Time. |
| uint8_t | oscillator_frequency | This byte represents the Slave's internal oscillator frequency in increments of 0.5 MHz (500kHz), up to 127.5 MHz. (See GETXTIME in the MIPI I3C Specification v1.1). |
| uint8_t | oscillator_inaccuracy | Oscillator inaccuracy in 0.5% increments of 0% up to 25.5% (See GETXTIME in the MIPI I3C Specification v1.1). |
| bool | hdr_ddr_support | HDR-DDR mode is supported. |
| bool | hdr_tsp_support | HDR-TSP mode is supported. |
| bool | hdr_tsl_support | HDR-TSL mode is supported. |

### ◆ i3c_instance_ctrl_t

| struct i3c_instance_ctrl_t |
|---|
| |

Channel control block. DO NOT INITIALIZE. Initialization occurs when i3c_api_t::open is called.

| **Public Member Functions** | |
|---|---|
| i3c_slave_info_t current_slave_info | BSP_ALIGN_VARIABLE (4) |
| | The last i3c_slave_info_t read during ENTDAA. |
| | |
| **Data Fields** | |
| uint32_t | open |
| | Indicates whether the open() API has been successfully called. |
| | |
| R_I3C0_Type * | p_reg |
| | Base register for this channel. |
| | |
| volatile uint32_t | internal_state |
| | Used to track the current state of the driver. |
| | |
| uint8_t | current_command_code |
| | The current Common Command Code that is being transferred. |
| | |
| uint32_t | device_index |
| | The device index selected using i3c_api_t::deviceSelect. |
| | |
| i3c_bitrate_mode_t | device_bitrate_mode |
| | Runtime bitrate settings to use for the next transfer. |
| | |
| uint32_t | next_word |
| | The next word that will be written to the FIFO. |
| | |
| uint32_t | ibi_next_word |

| | |
|---|---|
| | The next word that will be written to the IBI FIFO. |
| | |
| i3c_write_buffer_descriptor_t | write_buffer_descriptor |
| | Buffer descriptor for keeping track of a write transfer. |
| | |
| i3c_read_buffer_descriptor_t | read_buffer_descriptor |
| | Buffer descriptor for keeping track of a read transfer. |
| | |
| i3c_read_buffer_descriptor_t | ibi_buffer_descriptor |
| | Buffer descriptor for keeping track of an IBI read/write transfer. |
| | |
| volatile uint32_t | read_transfer_count_final |
| | The total number of bytes read during a read transfer. |
| | |
| volatile uint32_t | ibi_transfer_count_final |
| | The total number of bytes read during an IBI transfer. |
| | |
| i3c_cfg_t const * | p_cfg |
| | A pointer to the configuration structure provided during open. |
| | |

### ◆ i3c_extended_cfg_t

| struct i3c_extended_cfg_t | | |
|---|---|---|
| Extended configuration for r_i3c. | | |
| Data Fields | | |
| i3c_bitrate_settings_t | bitrate_settings | Bitrate settings configuring the frequency and duty cycle for SCL. |
| i3c_ibi_control_t | ibi_control | Configure the driver's behavior in response to IBIs. |
| uint32_t | bus_free_detection_time | The time in I3C reference clock |

| | | |
|---|---|---|
| | | ticks needed in order to detect the bus free condition (See "Bus Free Condition" in the MIPI I3C Specification v1.0). |
| uint32_t | bus_available_detection_time | The time in I3C reference clock ticks needed in order to detect the bus available condition (See "Bus Available Condition" in the MIPI I3C Specification v1.0). |
| uint32_t | bus_idle_detection_time | The time in I3C reference clock ticks needed in order to detect the bus idle condition (See "Bus Idle Condition" in the MIPI I3C Specification v1.0). |
| bool | timeout_detection_enable | Notify the application if SCL is stuck high or low. |
| i3c_slave_command_response_info_t | slave_command_response_info | Initial settings for configuring the slave's responses to received commands. |
| IRQn_Type | resp_irq | Response Queue Full IRQ number. |
| IRQn_Type | rx_irq | Receive FIFO Full IRQ number. |
| IRQn_Type | tx_irq | Transmit FIFO Empty IRQ number. |
| IRQn_Type | rcv_irq | Receive Status Queue Full IRQ number. |
| IRQn_Type | ibi_irq | IBI IRQ number. |
| IRQn_Type | eei_irq | EEI IRQ number. |
| uint8_t | ipl | Interrupt Priority for Resp, Rx, Tx, and RCV IRQs. |
| uint8_t | eei_ipl | Error and Event Interrupt Priority. |

### Macro Definition Documentation

#### ◆ I3C_ERROR_RECOVERY_VERSION_1

```
#define I3C_ERROR_RECOVERY_VERSION_1
```

Support error recovery procedure for chip version 1.

There are two different versions of the RA2E2 MCU and the error recovery procedure is different for each version.

### ◆ I3C_DEVICE_INDEX_EXTENDED_DEVICE

| #define I3C_DEVICE_INDEX_EXTENDED_DEVICE |
|---|
| Index for selecting the device defined in the extended address table. |

### ◆ I3C_EVENT_STATUS_SUCCESS

| #define I3C_EVENT_STATUS_SUCCESS |
|---|
| The transfer was completed as expected.<br>Event Status Provided by the callback. |

## Enumeration Type Documentation

### ◆ i3c_bitrate_mode_t

| enum i3c_bitrate_mode_t | |
|---|---|
| Bitrate settings that can be selected at run-time using i3c_api_t::deviceSelect. | |
| Enumerator | |
| I3C_BITRATE_MODE_I2C_STDBR | Use standard period setting for subsequent I2C transfers. |
| I3C_BITRATE_MODE_I2C_EXTBR | Use extended period setting for subsequent I2C transfers. |
| I3C_BITRATE_MODE_I3C_SDR0_STDBR | Use standard period setting for subsequent I3C SDR transfers. |
| I3C_BITRATE_MODE_I3C_SDR1_EXTBR | Use extended period setting for subsequent I3C SDR transfers. |
| I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2 | Use standard period setting x 2 for subsequent I3C SDR transfers. |
| I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2 | Use extended period setting x 2 for subsequent I3C SDR transfers. |
| I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4 | Use extended period setting x 4 for subsequent I3C SDR transfers. |
| I3C_BITRATE_MODE_I3C_HDR_DDR_STDBR | Use standard period setting for subsequent I3C HDR-DDR transfers. |

### ◆ i3c_activity_state_t

| enum i3c_activity_state_t | |
|---|---|
| Supported activity states for ENTASn Command (See ENTASn in the MIPI I3C Specification v1.0). | |
| Enumerator | |
| I3C_ACTIVITY_STATE_ENTAS0 | Activity Interval (1 microsecond). |
| I3C_ACTIVITY_STATE_ENTAS1 | Activity Interval (100 microseconds). |
| I3C_ACTIVITY_STATE_ENTAS2 | Activity Interval (2 milliseconds). |
| I3C_ACTIVITY_STATE_ENTAS3 | Activity Interval (50 milliseconds). |

### ◆ i3c_data_rate_setting_t

| enum i3c_data_rate_setting_t | |
|---|---|
| Maximum Sustained Data Rate for non-CCC messages sent by Master Device to Slave Device (See GETMXDS in the MIPI I3C Specification v1.0). | |
| Enumerator | |
| I3C_DATA_RATE_SETTING_FSCL_MAX | There is no data rate limit. |
| I3C_DATA_RATE_SETTING_8MHZ | The max sustained data rate is 8 Mhz. |
| I3C_DATA_RATE_SETTING_6MHZ | The max sustained data rate is 6 Mhz. |
| I3C_DATA_RATE_SETTING_4MHZ | The max sustained data rate is 4 Mhz. |
| I3C_DATA_RATE_SETTING_2MHZ | The max sustained data rate is 2 Mhz. |

### ◆ i3c_clock_data_turnaround_t

| enum i3c_clock_data_turnaround_t | |
|---|---|
| Clock to Data Turnaround Time (See GETMXDS in the MIPI I3C Specification v1.0). | |
| Enumerator | |
| I3C_CLOCK_DATA_TURNAROUND_8NS | Clock to turnaround time is 8 nanoseconds or less. |
| I3C_CLOCK_DATA_TURNAROUND_9NS | Clock to turnaround time is 9 nanoseconds or less. |
| I3C_CLOCK_DATA_TURNAROUND_10NS | Clock to turnaround time is 10 nanoseconds or less. |
| I3C_CLOCK_DATA_TURNAROUND_11NS | Clock to turnaround time is 11 nanoseconds or less. |
| I3C_CLOCK_DATA_TURNAROUND_12NS | Clock to turnaround time is 12 nanoseconds or less. |
| I3C_CLOCK_DATA_TURNAROUND_EXTENDED | Clock to turnaround time is greater than 12 nanoseconds. |

## Function Documentation

### ◆ R_I3C_Open()

fsp_err_t R_I3C_Open ( i3c_ctrl_t *const *p_api_ctrl*, i3c_cfg_t const *const *p_cfg* )

Configure an I3C instance. Implements i3c_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was invalid. |
| FSP_ERR_ALREADY_OPEN | Open has already been called for this instance. |
| FSP_ERR_UNSUPPORTED | A selected feature is not supported with the current configuration. |

### ◆ R_I3C_Enable()

fsp_err_t R_I3C_Enable ( i3c_ctrl_t *const *p_api_ctrl*)

Enable the I3C device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_INVALID_MODE | This instance is already enabled. |

### ◆ R_I3C_DeviceCfgSet()

fsp_err_t R_I3C_DeviceCfgSet ( i3c_ctrl_t *const *p_api_ctrl*, i3c_device_cfg_t const *const *p_device_cfg* )

Set the configuration for this device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_UNSUPPORTED | The device cannot be a secondary master if master support is disabled. |

### ◆ R_I3C_MasterDeviceTableSet()

fsp_err_t R_I3C_MasterDeviceTableSet ( i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, i3c_device_table_cfg_t const *const *p_device_table_cfg* )

Configure an entry in the master device table. This function is called in master mode in order to configure the devices on the I3C bus. It may also be called in slave mode when the slave receives the DEFSVLS command.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_UNSUPPORTED | Mastership requests must be rejected is slave support is disabled. |

### ◆ R_I3C_SlaveStatusSet()

fsp_err_t R_I3C_SlaveStatusSet ( i3c_ctrl_t *const *p_api_ctrl*, i3c_device_status_t *status* )

Set the status returned to the master in response to a GETSTATUS command.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_INVALID_MODE | The instance is not in slave mode. |
| FSP_ERR_UNSUPPORTED | Slave support is disabled. |

### ◆ R_I3C_DeviceSelect()

fsp_err_t R_I3C_DeviceSelect ( i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, uint32_t *bitrate_mode* )

In master mode, select the device for the next transfer. This function is not used in slave mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_INVALID_MODE | This operation is prohibited in slave mode. |
| FSP_ERR_UNSUPPORTED | Master support is disabled. |

### ◆ R_I3C_DynamicAddressAssignmentStart()

fsp_err_t R_I3C_DynamicAddressAssignmentStart ( i3c_ctrl_t *const *p_api_ctrl*, i3c_address_assignment_mode_t *address_assignment_mode*, uint32_t *starting_device_index*, uint32_t *device_count* )

Start the Dynamic Address Assignment Process. Implements i3c_api_t::dynamicAddressAssignmentStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_INVALID_MODE | This operation is prohibited in slave mode. |
| FSP_ERR_IN_USE | The operation could not be completed because the driver is busy. |
| FSP_ERR_UNSUPPORTED | Master support is disabled. |

### ◆ R_I3C_CommandSend()

fsp_err_t R_I3C_CommandSend ( i3c_ctrl_t *const *p_api_ctrl*, i3c_command_descriptor_t const *const *p_command_descriptor* )

Send a broadcast or direct command to slave devices on the bus. Implements i3c_api_t::commandSend.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_IN_USE | The operation could not be completed because the driver is busy. |
| FSP_ERR_INVALID_MODE | This driver is not in master mode. |
| FSP_ERR_INVALID_ALIGNMENT | The buffer must be aligned to 4 bytes. If it is a read operation, the length also be a multiple of 4 bytes. |
| FSP_ERR_UNSUPPORTED | Master support must be enabled to call this function. Slave support must be enabled when sending the GETACCMST command. |

### ◆ R_I3C_Write()

fsp_err_t R_I3C_Write ( i3c_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_data*, uint32_t *length*, bool *restart* )

Set the write buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_IN_USE | The operation could not be completed because the driver is busy. |
| FSP_ERR_INVALID_MODE | This driver is disabled, or an invalid bitrate mode is selected. |
| FSP_ERR_INVALID_ALIGNMENT | The buffer must be aligned to 4 bytes. |

### ◆ R_I3C_Read()

fsp_err_t R_I3C_Read ( i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length*, bool *restart* )

Set the read buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_IN_USE | The operation could not be completed because the driver is busy. |
| FSP_ERR_INVALID_MODE | This driver is disabled, or an invalid bitrate mode is selected. |
| FSP_ERR_INVALID_ALIGNMENT | The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes. |

### ◆ R_I3C_IbiWrite()

fsp_err_t R_I3C_IbiWrite ( i3c_ctrl_t *const *p_api_ctrl*, i3c_ibi_type_t *ibi_type*, uint8_t const *const *p_data*, uint32_t *length* )

Initiate an IBI write operation (This function is only used in slave mode).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_IN_USE | The operation could not be completed because the driver is busy. |
| FSP_ERR_INVALID_MODE | This function is only called in slave mode. |
| FSP_ERR_INVALID_ALIGNMENT | The buffer must be aligned to 4 bytes. |
| FSP_ERR_UNSUPPORTED | Slave supoprt is disabled. |

### ◆ R_I3C_IbiRead()

fsp_err_t R_I3C_IbiRead ( i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length* )

Set the read buffer for storing received IBI data (This function is only used in master mode).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |
| FSP_ERR_INVALID_MODE | This function is only called in master mode. |
| FSP_ERR_INVALID_ALIGNMENT | The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes. |
| FSP_ERR_UNSUPPORTED | Master support is disabled. |

### ◆ R_I3C_Close()

fsp_err_t R_I3C_Close ( i3c_ctrl_t *const *p_api_ctrl*)

Close the I3C instance. Implements i3c_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_NOT_OPEN | This instance has not been opened yet. |

## 5.2.6.17 LIN (r_sau_lin)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SAU_LIN_Open (lin_ctrl_t *const p_api_ctrl, lin_cfg_t const *const p_cfg) |
| fsp_err_t | R_SAU_LIN_Write (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SAU_LIN_Read (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |

| | |
|---|---|
| fsp_err_t | R_SAU_LIN_CommunicationAbort (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_LIN_CallbackSet (lin_ctrl_t *const p_api_ctrl, void(*p_callback)(lin_callback_args_t *), void *const p_context, lin_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SAU_LIN_WakeupSend (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_LIN_SleepEnter (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_LIN_SleepExit (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_LIN_Close (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_LIN_InformationFrameRead (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SAU_LIN_StartFrameWrite (lin_ctrl_t *const p_api_ctrl, uint8_t const id) |
| fsp_err_t | R_SAU_LIN_InformationFrameWrite (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |

### Detailed Description

LIN driver for the SAU peripheral on RA MCUs. This module implements the LIN Interface.

# Overview

SAU LIN supports Local Interface Network (LIN) transactions in master or slave mode. A callback must be provided, and is invoked when a transmission, reception, or other event has completed or occurred. The callback arguments contain information about the transaction status, bytes transferred and a pointer to the user-defined context.

The SAU LIN driver uses the SAU UART, TAU, and ICU peripherals to implement LIN communication.

### Features

- Half-duplex master or slave mode LIN communication
- Interrupt-driven data transmission and reception
- Optional LIN wake-up signal transmission and detection
- Generation of the sync (0x55) and Protected Identifier fields when in master mode
- Validation of the Protected Identifier field when in slave mode
- Optional checksum generation/validation using classic or enhanced LIN checksum
- Auto synchronization during sync byte reception is supported in slave mode
- Invoking the user-callback function with an event code (RX/TX complete, error, etc)
- Abort in-progress read/write operations
- Error notifications of parity error in protected identifier, framing error and overrun error
- Optional integration with the DTC transfer module

# Configuration

### Build Time Configurations for r_sau_lin

The following build time configurations are defined in fsp_cfg/r_sau_lin_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Checksum Support | • Enabled<br>• Disabled | Enabled | When set to 'Disabled', code for checksum generation and validation is excluded from the build. Disable only when checksum generation and validation is not required for any LIN instance. |
| Master Support | • Enabled<br>• Disabled | Disabled | Select whether code for master support is included in the build. |
| Slave Support | • Enabled<br>• Disabled | Enabled | Select whether code for slave support is included in the build. |
| Wake-up Support | • Enabled<br>• Disabled | Disabled | Select whether code for LIN bus sleep and wake-up signal detection is included in the build. Enable when at least one LIN instance requires sleep/wake-up support. |
| Auto Synchronization Support | • Enabled<br>• Disabled | Enabled | When set to 'Disabled', code for auto synchronization is excluded from the build. Enable when at least one LIN slave instance is using auto synchronization. |

### Configurations for Connectivity > LIN (r_sau_lin)

This module can be added to the Stacks tab via New Stack > Connectivity > LIN (r_sau_lin).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid | g_lin0 | Module name. |

| | | | |
|---|---|---|---|
| | C symbol | | |
| Mode | • Master<br>• Slave | Slave | Select the LIN mode. |
| Callback | Name must be a valid C symbol | sau_lin0_callback | A user callback function must be provided. It will be called from the interrupt service routine (ISR) in order to notify the application of LIN transmission/reception events and errors. |
| UART Channel | MCU Specific Options | | Select the LIN UART Channel. |
| TAU Channel | MCU Specific Options | | Select the LIN TAU Channel. |
| ICU Channel | MCU Specific Options | | Select the LIN ICU Channel. |
| Baud Rate | Value must be an integer greater than 0 | 19200 | Select the LIN Baudrate. |
| Auto Synchronization | • Disable<br>• Enable | Enable | Enable LIN synchronization to the master's clock by measuring pulse intervals between the falling edges of the input signal from the RXDn pin during reception of the LIN sync byte. |

**Clock Configuration**

In both master and slave modes, the SAU UART operation clock can be selected from the available operation clocks for the SAU unit.

In slave mode, the TAU operation clock can be selected from the available operation clocks for the TAU channel.

For more information about module clock settings, please refer to the clock setting of UART (r_sau_uart) and Independent Channel, 16-bit and 8-bit timer (r_tau).

**Pin Configuration**

This module uses TXD and RXD to communicate to external devices. Connect TXD and RXD to an on-board or external LIN transceiver for LIN bus communication.

An external pull up resistor is required on TXD in both master and slave modes.

**Dependent Module Configuration**

### SAU UART Module Configuration

- Disable 'Common > Enable Fixed Baudrate'
- Configure 'Interrupts > Transmit End Interrupt Priority' and 'Interrupts > Receive End Interrupt Priority'
- 'Interrupts > Error Interrupt Priority' is optional, but framing and overflow errors cannot be detected unless it is enabled.

### TAU Module Configuration

TAU is not used in master mode. In slave mode:

- Enable 'Common > Pin Input Support'
- Enable 'Common > Extra Input Mode Support'
- Configure 'Interrupts > Interrupt Priority'

### ICU Module Configuration

ICU is only required if using bus wake-up functionality (R_SAU_LIN_WakeupSend, R_SAU_LIN_SleepEnter, R_SAU_LIN_SleepExit).

- Configure ICU 'Pin Interrupt' priority when the ICU module is used.

## Usage Notes

### LIN Transmission and Reception

SAU LIN data reception is disabled until a LIN header is received (slave mode) or a call to R_SAU_LIN_Read() is made (master mode).

### Slave Mode Transmission and Reception

The header is detected in slave mode only. Header reception is interrupt driven, and is enabled after a successful call to R_SAU_LIN_Open() in slave mode.

When the LIN break field is detected by the slave node, header reception begins without action from the application. When header reception completes, the user callback is called.

If the slave node needs to receive a data frame, it must call R_SAU_LIN_Read() before the first data is received in order to receive the data. The user callback is called when reception is complete. It is permitted to call R_SAU_LIN_Read() from the callback context.

If the master node needs to publish a response to its own header, the master node must call R_SAU_LIN_Write() and provide a pointer to the response data. The response will be published following the header. The user callback is called when the full frame transmission is complete.

If the slave node neither needs to receive the data frame, nor publish a response for a received header, no action is required to ignore the frame.

### Master Mode Transmission and Reception

The header is not detected in master mode. Data frame reception is enabled only after a call to R_SAU_LIN_Read().

A header-only frame can be transmitted by calling R_SAU_LIN_Write() with a NULL data pointer. The user callback is called when transmission is complete.

If the master node needs to receive the data response, call R_SAU_LIN_Read() after header transmission has completed and before the first data is received. It is permitted to call R_SAU_LIN_Read() from the callback context.

If the master node needs to publish a response to its own header, the master node must call R_SAU_LIN_Write() with a non-NULL data pointer. The response will be published following the header. The user callback is called when the full frame transmission is complete.

### Sleep/Wake-up

### Sleep Mode Entry

R_SAU_LIN_SleepEnter can be called to place the bus into bus sleep mode. When in bus sleep mode, an external interrupt is enabled on the RXD pin which can be used to notify the application when bus activity on RXDn is detected.

R_SAU_LIN_SleepEnter does not place the MCU in a low power mode. To use it to wake from a low power mode, call R_SAU_LIN_SleepEnter, then use the Low Power Modes (r_lpm) driver to enter a low power mode.

### Sleep Mode Exit

Bus sleep mode can be exited in the following ways:

1. The LIN wake up signal is detected
2. The LIN break field is detected
3. The application calls R_SAU_LIN_SleepExit

*Note*

*If sleep mode is exited by break field detection, the incoming frame cannot be processed by the driver. Subsequent frames can be processed normally.*

### Wake-up Signal

Both the master and slave can send the LIN wake-up signal by calling R_SAU_LIN_WakeupSend.

Wake-up retry schedules, if required, must be implemented by the application.

If a LIN master receives the wake-up signal from a slave, the master should wait until the entire LIN cluster is ready before sending new header frames to ensure the frames can be processed by the slaves. The amount of time until the cluster is ready is application specific.

### Timeouts/Errors

The application is responsible for managing timeouts in case R_SAU_LIN_Read() is called, but the data is not received in the expected time period. If a timeout occurs, R_SAU_LIN_CommunicationAbort() may be called to cancel a pending read.

When an error occurs, reception is stopped and the user callback is called with the relevant error code.

### Checksum Generation and Validation

- The checksum calculation is performed in software for both checksum generation and validation.
- Checksum generation and validation are both optional.
- The LIN checksum may be generated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when writing data.
- The LIN checksum may be validated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when reading data.
- To skip driver checksum generation/validation, pass lin_transfer_params_t::LIN_CHECKSUM_TYPE_NONE to the read/write functions. Instead the checksum can be included as part of the data buffer, or omitted entirely depending on the application's needs.
- If not using driver checksum generation/validation for any LIN instance, it is recommended to set 'Checksum Support > Disabled' to reduce code size.
- The maximum number of data bytes that can be transmitted per frame is 255, including the checksum byte. Thus, when checksum generation/validation is requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 254. When checksum generation/validation is not requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 255.

**Auto Synchronization**

Auto synchronization is optional. When it is disabled, the baud rate set in R_SAU_LIN_Open is never updated.

This feature works in slave mode only.

- Auto synchronization is not the same as automatic baud rate detection. Auto synchronization is used for correcting small clock drift/difference in baud rate between nodes.
- Auto synchronization performs small automatic baud rate adjustments to synchronize with the master's clock by measuring pulse intervals between the falling edges of the input signal from the RXDn pin during reception of the LIN sync byte.
- The sync byte is measured and the baud rate is re-adjusted for every LIN sync byte received.
- When auto synchronization is enabled, it may change the operation clock frequency of the SAU unit during the baud rate adjustment. Since the operation clocks are shared between SAU channels on each SAU unit, altering the shared operation clock of one instance will result in an incorrect bitrate being generated on the other instance(s). Therefore, if your application cannot accommodate changes to the operation clock, it is advisable to disable auto synchronization or use a different operation clock for each instance.

**Limitations**

- Automatic (arbitrary) baud rate detection is not supported.
- The SAU provides an interface to the LIN Physical layer and does not implement the LIN Transport Layer or LIN Application Layer. The application is responsible for handling the frame scheduling table and monitoring for Transport Layer timeouts.
- A new LIN break field cannot be detected during slave sync byte measurement.

# Examples

### Basic LIN Slave Example

The following demonstrates a basic example of header reception and data response for a LIN slave.

```
void r_sau_lin_basic_slave_example (void)
{
 /* Initialize transmit buffer to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the header that this slave publishes the response to. */
 for ( ; ; )
    {
 if (g_header_rx_complete)
      {
 if (g_received_pid == FRAME_ID)
      {
 /* Frame ID of interest received */
 break;
      }
 /* Ignore this header. No action required by slave. */
           g_header_rx_complete = false;
      }
    }
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = g_received_pid,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data response */
```

```
    err = R_SAU_LIN_Write(&g_slave_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for the response transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SAU_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
void slave_callback (lin_callback_args_t * p_args)
{
 /* Handle the LIN event */
 switch (p_args->event)
    {
 case LIN_EVENT_RX_HEADER_COMPLETE:
     {
           g_header_rx_complete = 1;
           g_received_pid       = p_args->pid;
 break;
     }
 case LIN_EVENT_RX_DATA_COMPLETE:
     {
           g_data_rx_complete = 1;
 break;
     }
 case LIN_EVENT_TX_DATA_COMPLETE:
     {
           g_data_tx_complete = 1;
 break;
     }
 case LIN_EVENT_TX_WAKEUP_COMPLETE:
     {
           g_wakeup_tx_complete = 1;
```

```
 break;
      }
 case LIN_EVENT_RX_WAKEUP_COMPLETE:
      {
            g_wakeup_rx_complete = 1;
 break;
      }
 default:
      {
 /* Handle error */
      }
   }
}
```

### Basic LIN Master Read Example

The following demonstrates a basic example of frame reception for a LIN master.

```
void r_sau_lin_basic_master_read_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Send the header only */
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = 0,
       .p_data        = NULL          /* Set to NULL to send header only */
    };
    err = R_SAU_LIN_Write(&g_master_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for header transmission to complete before starting a read of the response
*/
```

```
while (!g_header_tx_complete)
   {
   }
lin_transfer_params_t read_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = FRAME_ID,
      .num_bytes     = TRANSFER_LENGTH,
      .p_data        = g_rx_buf
   };
/* Begin reception of the frame data */
   err = R_SAU_LIN_Read(&g_master_ctrl, &read_params);
   assert(FSP_SUCCESS == err);
/* Wait for data reception to complete. The application is responsible for
 * timing out if data is not received within the expected time interval*/
while (!g_data_rx_complete)
   {
   }
/* Close the driver */
   err = R_SAU_LIN_Close(&g_slave_ctrl);
   assert(FSP_SUCCESS == err);
}
void master_callback (lin_callback_args_t * p_args)
{
/* Handle the LIN event */
switch (p_args->event)
   {
case LIN_EVENT_RX_DATA_COMPLETE:
     {
           g_data_rx_complete = 1;
break;
     }
case LIN_EVENT_TX_HEADER_COMPLETE:
     {
```

```
            g_header_tx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_DATA_COMPLETE:
      {
            g_data_tx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_WAKEUP_COMPLETE:
      {
            g_wakeup_tx_complete = 1;
 break;
      }
 case LIN_EVENT_RX_WAKEUP_COMPLETE:
      {
            g_wakeup_rx_complete = 1;
 break;
      }
 case LIN_EVENT_ERR_INVALID_CHECKSUM:
 case LIN_EVENT_ERR_BUS_COLLISION_DETECTED:
 case LIN_EVENT_ERR_FRAMING:
 case LIN_EVENT_ERR_COUNTER_OVERFLOW:
 case LIN_EVENT_ERR_OVERRUN:
 case LIN_EVENT_ERR_PARITY:
 default:
      {
 /* Handle error */
      }
    }
}
```

### Basic LIN Master Write Example

The following demonstrates a basic example of frame transmission for a LIN master.

```
void r_sau_lin_basic_master_write_example (void)
{
 /* Initialize transmit buffer to known data. */
 for (uint8_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
    err = R_SAU_LIN_Write(&g_master_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for data transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SAU_LIN_Close(&g_master_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Basic LIN Send Wake Up Example

The following demonstrates a basic example of sending wake up signal for a LIN master. The usage of slave mode is the same.

```
void r_sau_lin_send_wakeup_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Send the LIN wake up signal */
    err = R_SAU_LIN_WakeupSend(&g_master_ctrl);
    assert(FSP_SUCCESS == err);
 while (!g_wakeup_tx_complete)
    {
    }
 /* Application specific delay for slaves to re-initialize */
 R_BSP_SoftwareDelay(SLAVE_INITIALIZATION_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Send additional frames as needed */
 /* Close the driver */
    err = R_SAU_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Basic LIN Sleep Enter Example

The following demonstrates a basic example of entering sleep mode for a LIN slave. The usage in master mode is the same.

```
void r_sau_lin_sleep_enter_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* Open IRQ to prepare getting LIN_EVENT_RX_WAKEUP_COMPLETE */
    err = R_SAU_LIN_SleepEnter(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
 /* If a low power mode is required, call R_LPM_LowPowerModeEnter */
 while (!g_wakeup_rx_complete)
    {
```

```
    }
 /* Close the driver */
    err = R_SAU_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Basic LIN Sleep Exit Example

The following demonstrates a basic example of exiting the LIN slave sleep mode. The usage in master mode is the same.

```
void r_sau_lin_sleep_exit_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SAU_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* Enter bus sleep mode */
    err = R_SAU_LIN_SleepEnter(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
 /* Exit the bus sleep mode for LIN device.*/
    err = R_SAU_LIN_SleepExit(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_SAU_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | sau_lin_extended_cfg_t |
| struct | sau_lin_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sau_lin_state_t |
| enum | sau_lin_synchronization_t |

## Data Structure Documentation

### ◆ sau_lin_extended_cfg_t

| struct sau_lin_extended_cfg_t | | |
|---|---|---|
| SAU UART LIN extended configuration | | |
| Data Fields | | |
| const uart_instance_t * | p_uart | The UART instance used in SAU LIN. |
| const external_irq_instance_t * | p_icu | The IRQ instance used in SAU LIN. |

### ◆ sau_lin_instance_ctrl_t

| struct sau_lin_instance_ctrl_t |
|---|
| LIN Instance Control Block. DO NOT INITIALIZE. |

## Enumeration Type Documentation

### ◆ sau_lin_state_t

| enum sau_lin_state_t | |
|---|---|
| SAU LIN Event codes | |
| Enumerator | |
| SAU_LIN_STATE_NONE | Master mode: No event present Slave mode: Waiting for break field signal. |
| SAU_LIN_STATE_SENDING_WAKEUP | Transmitting wakeup signal. |
| SAU_LIN_STATE_SENDING_BREAK_FIELD | Transmitting break field signal. |
| SAU_LIN_STATE_SENDING_SYNC | Transmitting sync word. |
| SAU_LIN_STATE_AWAITING_SYNC_SIGNAL | Waiting to receive sync word. |
| SAU_LIN_STATE_SENDING_PID | Transmitting protected identifier. |
| SAU_LIN_STATE_AWAITING_PID_SIGNAL | Waiting to receive protected identifier. |
| SAU_LIN_STATE_HEADER_RECEIVED | LIN header received. |
| SAU_LIN_STATE_SENDING_DATA | Transmitting LIN data. |
| SAU_LIN_STATE_AWAITING_DATA | Receiving LIN data. |
| SAU_LIN_STATE_AWAITING_CHECKSUM | Receiving LIN checksum. |
| SAU_LIN_STATE_BUS_SLEEP | Bus sleep state. |

### ◆ sau_lin_synchronization_t

| enum sau_lin_synchronization_t | |
|---|---|
| Auto synchronization setting. | |
| Enumerator | |
| SAU_LIN_AUTO_SYNCHRONIZATION_DISABLE | Disable auto synchronization during sync byte reception. |
| SAU_LIN_AUTO_SYNCHRONIZATION_ENABLE | Enable auto synchronization during sync byte reception. |

## Function Documentation

### ◆ R_SAU_LIN_Open()

fsp_err_t R_SAU_LIN_Open ( lin_ctrl_t *const *p_api_ctrl*, lin_cfg_t const *const *p_cfg* )

Configures the LIN driver channel based on the input configuration.

Implements lin_api_t::open.

Example:

```
/* Open the LIN instance with initial configuration. */
fsp_err_t err = R_SAU_LIN_Open(&g_master_ctrl, &g_master_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, configuration structure, or required dependency is NULL |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_MODE | Setting not supported for selected mode |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_Write()

fsp_err_t R_SAU_LIN_Write ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking transmission of a LIN frame.

In master mode, the LIN header (break, sync and protected identifier) is sent before the LIN data.

- To transmit the LIN header only, set p_transfer_params->p_data to NULL.
- To transmit both header and data, set p_transfer_params->p_data to the TX buffer to transmit.

The LIN header is not transmitted in slave mode. Set p_transfer_params->p_data to the TX buffer to transmit.

On successful header-only transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_HEADER_COMPLETE. On successful data transmission or header+data transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_DATA_COMPLETE.

Implements lin_api_t::write.

Example:

```
lin_transfer_params_t write_params =
   {
```

```
        .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
        .id            = FRAME_ID,
        .num_bytes     = TRANSFER_LENGTH,
        .p_data        = g_tx_buf
    };
 /* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
    err = R_SAU_LIN_Write(&g_master_ctrl, &write_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission started successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, transfer parameters, or tx/rx buffer is NULL, or 0 bytes length provided |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SAU_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_Read()

fsp_err_t R_SAU_LIN_Read ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking data reception to receive user specified number of bytes into destination buffer pointer.

The checksum type specifies the checksum type used for validation. If a non-standard algorithm is used, or the application prefers to validate the checksum outside the driver, or the application prefers to skip checksum validation, specify lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE. If checksum validation is skipped, the lin_checksum_type_t::LIN_EVENT_ERR_INVALID_CHECKSUM event is not possible. When lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE is used, the number of bytes specified in the receive buffer length will be received (the driver will not expect to receive an additional 1 checksum byte), so if a non-standard checksum is used, sufficient space must be allocated in the write buffer and accounted for in the provided length.

On successful data reception, the callback is called with event lin_event_t::LIN_EVENT_RX_DATA_COMPLETE.

Implements lin_api_t::read.

Example:

```
/* Begin reception of the frame data */
   err = R_SAU_LIN_Read(&g_master_ctrl, &read_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception started successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, transfer parameters, or tx/rx buffer is NULL, or 0 bytes length provided, no header matching pid was received(only for slave mode). |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SAU_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |
| FSP_ERR_INVALID_CALL | Data reception is not possible because a header frame has not been received yet (slave mode only). |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_CommunicationAbort()

fsp_err_t R_SAU_LIN_CommunicationAbort ( lin_ctrl_t *const *p_api_ctrl*)

Cancels in progress data read or write, or header read or write. Break field reception cannot be cancelled. For slave nodes, reception of a new header reception is still enabled after a call to this function.

Implements lin_api_t::communicationAbort.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transfer aborted successfully or no transfer was in progress. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_INVALID_STATE | Driver is in bus sleep state. Call R_SAU_LIN_SleepExit to return to active state or wait for wakeup signal. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_CallbackSet()

fsp_err_t R_SAU_LIN_CallbackSet ( lin_ctrl_t *const *p_api_ctrl*, void(*)(lin_callback_args_t *) *p_callback*, void *const *p_context*, lin_callback_args_t *const *p_callback_memory* )

Updates the user callback and callback context.

***Note***

*Providing callback memory is not supported, pass NULL to this function for p_callback_memory.*

Implements lin_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, callback function is NULL, or p_callback_memory is not NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_STATE | Driver is in bus sleep state. Call R_SAU_LIN_SleepExit to return to active state or wait for wakeup signal. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_WakeupSend()

fsp_err_t R_SAU_LIN_WakeupSend ( lin_ctrl_t *const *p_api_ctrl*)

Begins non-blocking transmission of the LIN wakeup signal.

On successful wakeup signal transmission, the callback is called with event lin_event_t::LIN_EVENT_TX_WAKEUP_COMPLETE.

Implements lin_api_t::wakeupSend.

Example:

```
/* Send the LIN wake up signal */
   err = R_SAU_LIN_WakeupSend(&g_master_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Wakeup signal transmission successfully started. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | Unsupported when Wake-up Support build option is disabled. |
| FSP_ERR_INVALID_STATE | Driver is in bus sleep state. Call R_SAU_LIN_SleepExit to return to active state or wait for wakeup signal. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_SleepEnter()

fsp_err_t R_SAU_LIN_SleepEnter ( lin_ctrl_t *const *p_api_ctrl*)

Places the LIN node in bus sleep mode. Enables the external IRQ that can be used to detect the LIN wakeup signal.

On wakeup signal reception, the callback is called with event lin_event_t::LIN_EVENT_RX_WAKEUP_COMPLETE and bus sleep mode is automatically exited.

***Note***

*This function does not put the MCU into a low power mode. Call this function prior to entering a low power mode to enable LIN wakeup signal detection support.*

Implements lin_api_t::sleepEnter.

Example:

```
/* Open IRQ to prepare getting LIN_EVENT_RX_WAKEUP_COMPLETE */
   err = R_SAU_LIN_SleepEnter(&g_slave_ctrl);
   assert(FSP_SUCCESS == err);
/* If a low power mode is required, call R_LPM_LowPowerModeEnter */
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Bus successfully transitioned to sleep state. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | Unsupported when Wake-up Support build option is disabled. |
| FSP_ERR_INVALID_STATE | Driver is in bus sleep state. Call R_SAU_LIN_SleepExit to return to active state or wait for wakeup signal. |
| FSP_ERR_NOT_ENABLED | ICU module is not enabled. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_SleepExit()

fsp_err_t R_SAU_LIN_SleepExit ( lin_ctrl_t *const *p_api_ctrl*)

Exits the bus sleep mode for LIN device. Disables wakeup signal detection.

Implements lin_api_t::sleepExit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Bus successfully transitioned to active state |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | Unsupported when Wake-up Support build option is disabled. |
| FSP_ERR_NOT_ENABLED | ICU module is not enabled. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_Close()

fsp_err_t R_SAU_LIN_Close ( lin_ctrl_t *const *p_api_ctrl*)

Closes the LIN driver.

Implements lin_api_t::close.

Example:

```
/* Close the driver */
   err = R_SAU_LIN_Close(&g_master_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_InformationFrameRead()

fsp_err_t R_SAU_LIN_InformationFrameRead ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* )

DEPRECATED. Use R_SAU_LIN_Read.

**Returns**
See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_StartFrameWrite()

fsp_err_t R_SAU_LIN_StartFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, uint8_t const *id* )

[DEPRECATED] Use R_SAU_LIN_Write

**Returns**
See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SAU_LIN_InformationFrameWrite()

fsp_err_t R_SAU_LIN_InformationFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* )

[DEPRECATED] Use R_SAU_LIN_Write

**Returns**
See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.6.18 LIN (r_sci_b_lin)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_B_LIN_Open (lin_ctrl_t *const p_api_ctrl, lin_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_B_LIN_Write (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_B_LIN_Read (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_B_LIN_CommunicationAbort (lin_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | R_SCI_B_LIN_CallbackSet (lin_ctrl_t *const p_api_ctrl, void(*p_callback)(lin_callback_args_t *), void *const p_context, lin_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_B_LIN_BaudCalculate (sci_b_lin_baud_params_t const *const p_baud_params, sci_b_lin_baud_setting_t *const p_baud_setting) |
| fsp_err_t | R_SCI_B_LIN_IdFilterSet (lin_ctrl_t *const p_api_ctrl, sci_b_lin_id_filter_setting_t const *const p_config) |
| fsp_err_t | R_SCI_B_LIN_IdFilterGet (lin_ctrl_t *const p_api_ctrl, sci_b_lin_id_filter_setting_t *const p_config) |
| fsp_err_t | R_SCI_B_LIN_WakeupSend (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_LIN_SleepEnter (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_LIN_SleepExit (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_LIN_Close (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_LIN_InformationFrameRead (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_B_LIN_StartFrameWrite (lin_ctrl_t *const p_api_ctrl, uint8_t const id) |
| fsp_err_t | R_SCI_B_LIN_InformationFrameWrite (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the LIN Interface.

# Overview

The Simple LIN on SCI_B HAL module supports Local Interface Network (LIN) transactions in master or slave mode. A callback must be provided, and is invoked when a transmission, reception, or other event has completed or occurred. The callback arguments contain information about the transaction status, bytes transferred and a pointer to the user defined context.

### Features

- Half-duplex master or slave mode LIN communication
- Interrupt-driven data transmission and reception
- Generation of the sync (0x55) and Protected Identifier fields when in master mode
- Validation of the Protected Identifier field when in slave mode
- ID filtering support in slave mode
- Optional checksum generation/validation using classic or enhanced LIN checksum

- Invoking the user-callback function with an event code (RX/TX complete, error, etc)
- Auto synchronization is supported in slave mode
- Adjustable break field length
- Adjustable break field delimiter/stop bits length
- Noise cancellation
- Abort in-progress read/write operations
- Error notifications of parity error in protected identifier, framing error, overrun error, bus collision, and counter overflow error (counter overflow error applies in slave mode only)
- Operation clock selection (PCLK or SCISPI/SCICLK)
- Configuration of all available SCI_B channels with Simple LIN mode support

# Configuration

### Build Time Configurations for r_sci_b_lin

The following build time configurations are defined in fsp_cfg/r_sci_b_lin_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Checksum Support | • Enabled<br>• Disabled | Enabled | When set to 'Disabled', code for checksum generation and validation is excluded from the build. This setting is applied globally to the project. Disable only when checksum generation and validation is not required for any LIN instance. |
| Auto Synchronization Support | • Enabled<br>• Disabled | Disabled | When set to 'Disabled', code for auto synchronization is excluded from the build. This setting is applied globally to the project. Enable when at least one LIN slave instance is using auto synchronization. |

### Configurations for Connectivity > LIN (r_sci_b_lin)

This module can be added to the Stacks tab via New Stack > Connectivity > LIN (r_sci_b_lin). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_lin0 | Module name. |
| Channel | MCU Specific Options | | Select the LIN channel. |
| Mode | • Master<br>• Slave | Master | Select the LIN operating mode (master or slave). |
| Extra | | | |
| Clock Source | MCU Specific Options | | Select whether the peripheral clock (PCLK) or SCICLK/SCISPICLK is used as the baud rate generator. |
| Noise Filter | • Enable<br>• Disable | Disable | Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits. |
| Noise Filter Clock Select | • The base clock signal divided by 1<br>• The on-chip baud rate generator source clock divided by 1<br>• The on-chip baud rate generator source clock divided by 2<br>• The on-chip baud rate generator source clock divided by 4<br>• The on-chip baud rate generator source clock divided by 8 | The base clock signal divided by 1 | Select the clock division setting for the digital noise filter |
| Bus Conflict Detection | • Enable<br>• Disable | Disable | Enable the bus conflict detection function. When enabled, the TXDn pin output and the RXDn pin input are sampled by the bus conflict detection clock set by Extra\|Bus Conflict Clock Divider. |

| | | | |
|---|---|---|---|
| | | | This function only works during transmission. |
| Bus Conflict Clock Divider | • 1<br>• 2<br>• 4 | 2 | Select the base clock divider for the sampling clock of the bus conflict detection circuit. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 19200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated sci_b_lin_extended_cfg_t structure. |
| Auto Synchronization | • Disable<br>• Enable | Disable | Enable LIN synchronization to the master's clock by measuring the effective edges of the input signal from the RXDn pin during reception of the sync byte. |
| Base Clock Cycles Per Bit Period | • Auto-Select<br>• 8<br>• 16 | Auto-Select | Select number of SCI base clock cycles per single bit period at the configured baud rate. When 16 base clock cycles is selected, the period of the SCI base clock is 1/16 of a single bit period. When 8 base clock cycles is selected, the period of the SCI base clock is 1/8 of a single bit period. 'Auto-Select' will calculate the base clock period automatically. 'Auto-Select' is not recommended when bus conflict detection is |

| | | | |
|---|---|---|---|
| | | | used, see usage notes. |
| Framing | | | |
| Framing > ID Filter (Slave Mode) | | | |
| Compare Data Mask | Value must be an integer between 0 and 255. | 0 | Select the bit mask to be applied before comparing the received PID to the selected compare data. This setting specifies which bits of the selected compare data must match. Set to 0 to disable the filter (allow all frame identifiers). |
| Priority Compare Data | Value must be an integer between 0 and 255. | 0 | Select the priority compare filter data. |
| Secondary Compare Data | Value must be an integer between 0 and 255. | 0 | Select the secondary compare filter data. |
| Compare Data Select | • Priority<br>• Secondary<br>• Both | Priority | Select the compare data to use. If 'both' is selected, the priority compare data is checked before the secondary compare data. |
| Priority Interrupt Bit Enable | • Enabled<br>• Disabled | Disabled | Select whether to enable the Priority Interrupt Bit filter. When enabled, regardless of the Filter Data Select setting value, the bit specified by the Priority Interrupt Bit setting is compared with the corresponding bit in the Priority Compare Data Filter and if it matches, the identifier is allowed through the filter. |
| Priority Interrupt Bit | Value must be an integer between bit 0 (LSb) and bit 7 (MSb). | 0 | Select the priority interrupt bit (0-7) of the received PID to compare with the corresponding bit in the Priority Compare Data Filter. |

| | | | |
|---|---|---|---|
| Break Field Bits/Break Detection Threshold (bits) | Value must be an integer. | 13 | For master break field transmission, this configures the dominant period of the break field (in bits). For slave break field reception, this configures the break detection threshold in bits. Must be 13 bits or greater for master mode and 11 bits or greater for slave mode. |
| LIN Timer Divider | • 4<br>• 16<br>• 64 | 4 | Set the LIN timer divider. The LIN timer is used for break field transmission and detection. Higher dividers make possible transmission/detection of more break field bits, but are less precise. |
| Break Field Delimiter/Stop Bits | • 1bit<br>• 2bits | 1bit | Select the recessive period (in bits) of the break field. This setting also selects the number of stop bits. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | sci_b_lin0_callback | A user callback function must be provided. It will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Data Empty Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |
| Break Field Detection Interrupt Priority | MCU Specific Options | | Select the break field detection interrupt priority (slave mode only). |
| Active Edge Detection Interrupt Priority | MCU Specific Options | | Select the active edge detection interrupt |

priority.

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA6T2 | SCISPICLK |
| RA8D1 | SCICLK |
| RA8D2 | SCICLK |
| RA8E1 | SCICLK |
| RA8E2 | SCICLK |
| RA8M1 | SCICLK |
| RA8M2 | SCICLK |
| RA8P1 | SCICLK |
| RA8T1 | SCICLK |
| RA8T2 | SCICLK |

The clock source for the baud-rate clock generator can be selected from either SCISCPI/SCICLK or the peripheral clock (PCLK).

*Note*

*1. See Figure 26.2 "Clock source selector block diagram" in the RA6T2 manual for more information.*

### Pin Configuration

This module uses TXD and RXD to communicate to external devices. Connect TXD and RXD to an on-board or external LIN transceiver for LIN bus communication.

An external pull up resistor is required on TXD in both master and slave mode.

### Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI), transmit end (TEI), and error (ERI) interrupts for the selected channel used must be enabled in the properties of the selected device for both master and slave mode operation.
- Break field detect (BFD) must be enabled in the properties of the selected device for slave mode only.

### Break Field Length/Detection Threshold Configuration

- When in master mode, this setting configures the low (dominant) time of the LIN break pattern.
- When in slave mode, this setting configures the break field detection threshold (number of bits that must be received to detect a break field).
- When configuring this setting in slave mode, take care to set the detection threshold less than the master break field length.

- The standard settings are 13 bits for master nodes and 11 bits for slave nodes, but alternate settings are supported.

**Bus Conflict Detection Configuration**

When Bus Conflict Detection is enabled, the TXDn pin output and the RXDn pin input are sampled by the selected bus conflict detection clock. When a mismatch occurs three times in a row, an SCIn_ERI interrupt is generated and transmission is stopped. This function only works during transmission.

Consideration of the specific application's LIN bus characteristics should be taken into account when enabling this function. If the bus collision sampling clock is too fast, the expected propagation delay through the transceiver may be erroneously detected as a bus collision (false positive). If the bus collision sampling clock is too slow, collisions may be missed (false negative).

When CCR2.ABCS = 1, setting the bus conflict clock divider to 4 is is prohibited by the SCI_B hardware.

A LIN transceiver is required to detect bus collisions. The propagation delay through the transceiver varies depending on individual transceiver design.

When Bus Conflict Detection is enabled, set the Bus Conflict Clock Divider to divide down the clock at least slow enough to avoid false positive collision detections caused by the worst case transceiver propagation delay. The Baud|Base Clock Cycles Per Bit Period setting can be used to adjust the SCI_B Base Clock frequency if needed.

***Example:***

The SCI_B base clock frequency can be calculated by the following formula:

sci_base_clock_frequency = 1 / [ (1/baud_rate)/cycles_per_bit_period ]

The following relationship must be satisfied to avoid false positives:

2 * [ 1e6 / (sci_base_clock_frequency/bus_conflict_clock_divider) ] > max_propagation_delay_us

Suppose:

- Transceiver TXD->LIN BUS max delay = 6us
- Transceiver LIN BUS->RXD max delay = 6us
- Transceiver max_propagation_delay_us: 6us + 6us = 12us
- baud_rate = 19200
- cycles_per_bit_period = 16

Then:

- 1 bit time @ 19200bps = 1e6/19200 = 52us
- sci_base_clock_frequency = 1/[ (1/19200)/16 ] = 307200 Hz

Bus Conflict Divider Selection Process:

- Divider of 1:
  - 2 * [ 1e6 / (307200/1) ] = 6.51 us (not > 12us, may cause false positives)
- Divider of 2:
  - 2 * [ 1e6 / (307200/2) ] = 13.02 us (is > 12us, can be used)
- Divider of 4:
  - 2 * [ 1e6 / (307200/4) ] = 26.04 us (is > 12us, can be used, but will be less accurate at detecting real collisions than divider of 2 due to slower bus collision clock)

If no possible divider is found with 16 cycles, Baud|Base Clock Cycles Per Bit Period can be reduced to 8, which will reduce the SCI_B Base Clock by half compared to 16.

If Baud|Base Clock Cycles Per Bit Period is set to 'Auto Select' the cycles per bit period will be selected automatically. In slave mode, if auto synchronization is enabled, the base cycles per bit period may change during bit rate synchronization, but the bus conflict clock divider and noise filter clock divider will not be changed at runtime. To prevent changing the base cycles per bit period during bit rate synchronization, fix the base clock cycles to 16 or 8 cycles by the procedure described above.

### ID Filter Configuration

The SCI performs optional hardware PID filtering in slave mode. When enabled, only headers with PIDs that pass through the filter are received and passed to the user callback. Other headers are ignored.

An initial configuration of the filter settings can be applied in the properties view by selecting Framing -> ID Filter and choosing the desired settings. See sci_b_lin_id_filter_setting_t for the available filter settings, as well as the examples in this document. By default, filtering is disabled and all PIDs can be received.

The filter can be updated at runtime by using using R_SCI_B_LIN_IdFilterSet. The current filter settings can be viewed with R_SCI_B_LIN_IdFilterGet.

## Usage Notes

### LIN Transmission and Reception

SCI data reception is disabled until a break field is detected (slave mode) or a call to R_SCI_B_LIN_Read() is made (master and slave mode).

### Slave Mode Transmission and Reception

The header is detected in slave mode only. Header reception is interrupt driven, and is enabled after a successful call to R_SCI_B_LIN_Open() in slave mode.

When the LIN break field is detected by the slave node, header reception begins without action from the application. When header reception completes, the user callback is called.

If the slave node needs to receive the data frame, the slave node must call R_SCI_B_LIN_Read() before the first data is received in order to receive the data. The user callback is called when reception is complete. It is permitted to call R_SCI_B_LIN_Read() from the callback context.

If the slave node needs to publish a response to the header, the slave node must call R_SCI_B_LIN_Write() and provide the response data and the PID of the received header in the lin_transfer_params_t. The user callback is called when transmission is complete.

If the slave node neither needs to receive a data frame, nor publish a response for a received header, no action is required to ignore the frame. Data reception is disabled until a new break field is detected.

**Master Mode Transmission and Reception**

The header is not detected in master mode. Data frame reception is enabled only after a call to R_SCI_B_LIN_Read().

The header is transmitted by calling R_SCI_B_LIN_Write(). The user callback is called when transmission is complete.

If the master node needs to receive the data response, call R_SCI_B_LIN_Read() after header transmission has completed and before the first data is received. It is permitted to call R_SCI_B_LIN_Read() from the callback context.

If the master node needs to publish a response to its own header, the master node must call R_SAU_LIN_Write() and provide a pointer to the response data. The response will be published following the header. The user callback is called when the full frame transmission is complete.

**Timeouts/Errors**

The application is responsible for managing timeouts in case R_SCI_B_LIN_Read() is called, but the data is not received in the expected time period. If a timeout occurs, R_SCI_B_LIN_CommunicationAbort() may be called to cancel a pending read.

When an error occurs, reception is stopped and the user callback is called with the relevant error code.

**Checksum Generation and Validation**

- The checksum calculation is performed in software for both checksum generation and validation.
- Checksum generation and validation are both optional.
- The LIN checksum may be generated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when writing data.
- The LIN checksum may be validated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when reading data.
- To skip driver checksum generation/validation, pass lin_transfer_params_t::LIN_CHECKSUM_TYPE_NONE to the read/write functions. Instead the checksum can be included as part of the data buffer, or omitted entirely depending on the application's needs.
- The maximum number of data bytes that can be transmitted per frame is 255, including the checksum byte. Thus, when checksum generation/validation is requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 254. When checksum generation/validation is not requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 255.

### Auto Synchronization

Auto synchronization is optional. When it is disabled, the baud rate and break field detection threshold set in open are never updated.

This feature works in slave mode only.

- Auto synchronization is not the same as automatic baud rate detection. Auto synchronization is used for correcting small clock drift/difference in baud rate between nodes.
- Auto synchronization performs small automatic baud rate adjustments to synchronize with the master's clock by measuring the effective edges of the input signal from the RXDn pin during reception of the LIN sync byte.
- The sync byte is measured and the baud setting and break field detection threshold are re-adjusted for every LIN sync byte received.
- If reception of the sync byte fails, or the required register settings detected would not be possible in the hardware, the baud/timer settings are not updated, and auto synchronization will be re-attempted on the next received header.
- If two or more headers are received consecutively and ignored by software (no call to R_SCI_B_LIN_Read), *and* there is no data transmission by any node on the bus between the headers, auto synchronization is performed only on the first header. Following the transmission of the next data frame (by any node), auto synchronization will be re-attempted on the next received header.

### Limitations

- DTC/DMAC is not supported
- Automatic baud rate detection is not supported
- The SCI_B provides an interface to the LIN Physical layer and does not implement the LIN Transport Layer or LIN Application Layer. The application is responsible for handling the frame scheduling table and monitoring for Transport Layer timeouts.
- LIN wake-up via external IRQ is not supported by the SCI_B (R_SCI_B_LIN_SleepEnter, R_SCI_B_LIN_SleepExit and R_SCI_B_LIN_WakeupSend are not supported)

## Examples

### Basic LIN Master Read Example

The following demonstrates a basic example of frame reception for a LIN master.

```
void r_sci_b_lin_basic_master_read_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_B_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Send the header only */
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
```

```
      .id            = FRAME_ID,
      .num_bytes     = 0,
      .p_data        = NULL          /* Set to NULL to send header only */
   };
   err = R_SCI_B_LIN_Write(&g_master_ctrl, &write_params);
   assert(FSP_SUCCESS == err);
 /* Wait for header transmission to complete before starting a read of the response
*/
 while (!g_header_tx_complete)
   {
   }
 lin_transfer_params_t read_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = FRAME_ID,
      .num_bytes     = TRANSFER_LENGTH,
      .p_data        = g_rx_buf
   };
 /* Begin reception of the data */
   err = R_SCI_B_LIN_Read(&g_master_ctrl, &read_params);
   assert(FSP_SUCCESS == err);
 /* Wait for data reception to complete. The application is responsible for
  * timing out if data is not received within the expected time interval*/
 while (!g_data_rx_complete)
   {
   }
 /* Close the driver */
   err = R_SCI_B_LIN_Close(&g_slave_ctrl);
   assert(FSP_SUCCESS == err);
}
void master_callback (lin_callback_args_t * p_args)
{
 /* Handle the LIN event */
 switch (p_args->event)
```

```
    {
 case LIN_EVENT_RX_DATA_COMPLETE:
      {
            g_data_rx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_HEADER_COMPLETE:
      {
            g_header_tx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_DATA_COMPLETE:
      {
            g_data_tx_complete = 1;
 break;
      }
 case LIN_EVENT_ERR_INVALID_CHECKSUM:
 case LIN_EVENT_ERR_BUS_COLLISION_DETECTED:
 case LIN_EVENT_ERR_FRAMING:
 case LIN_EVENT_ERR_COUNTER_OVERFLOW:
 case LIN_EVENT_ERR_OVERRUN:
 case LIN_EVENT_ERR_PARITY:
 default:
      {
 /* Handle error */
      }
    }
}
```

### Basic LIN Master Write Example

The following demonstrates a basic example of frame transmission for a LIN master.

```
void r_sci_b_lin_basic_master_write_example (void)
{
```

```
 /* Initialize transmit buffer to known data. */
 for (uint8_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_B_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
    err = R_SCI_B_LIN_Write(&g_master_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_B_LIN_Close(&g_master_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Basic LIN Slave Example

The following demonstrates a basic example of header reception and data response for a LIN slave.

```
void r_sci_b_lin_basic_slave_example (void)
{
 /* Initialize transmit buffer to known data */
```

```
for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
   {
      g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
   }
/* Open the LIN instance with initial configuration. */
fsp_err_t err = R_SCI_B_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
   assert(FSP_SUCCESS == err);
/* Wait for the header that this slave publishes the response to. */
for ( ; ; )
   {
if (g_header_rx_complete)
     {
if (g_received_pid == FRAME_ID)
     {
/* Frame ID of interest received */
break;
     }
/* Ignore this header. No action required by slave. */
           g_header_rx_complete = false;
     }
   }
lin_transfer_params_t write_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = g_received_pid,
      .num_bytes    = TRANSFER_LENGTH,
      .p_data       = g_tx_buf
   };
/* Send the LIN data response */
   err = R_SCI_B_LIN_Write(&g_slave_ctrl, &write_params);
   assert(FSP_SUCCESS == err);
/* Wait for the response transmission to complete */
while (!g_data_tx_complete)
   {
```

```
    }
 /* Close the driver */
    err = R_SCI_B_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
void slave_callback (lin_callback_args_t * p_args)
{
 /* Handle the LIN event */
 switch (p_args->event)
    {
 case LIN_EVENT_RX_HEADER_COMPLETE:
      {
            g_header_rx_complete = 1;
            g_received_pid       = p_args->pid;
 break;
      }
 case LIN_EVENT_RX_DATA_COMPLETE:
      {
            g_data_rx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_DATA_COMPLETE:
      {
            g_data_tx_complete = 1;
 break;
      }
 default:
      {
 /* Handle error */
      }
    }
}
```

### Basic LIN Slave ID Filtering Example

The following example demonstrates a basic example of frame ID filtering.

```
#define BASIC_FILTER_MASK (0x30)
#define BASIC_PRIORITY_FILTER_DATA (0x20)
#define BASIC_SECONDARY_FILTER_DATA (0x00) // Don't-care value in this example
#define BASIC_PRIORITY_INTERRUPT_BIT (0) // Don't-care value in this example
void r_sci_b_lin_basic_id_filtering_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_B_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* If the application is using dynamically assigned identifiers, perform
  * any required initial communications to get the identifiers for this slave */
 /* Configure filter to accept only frame IDs in the range 0x20-0x2F.
  *
  * The header is received only when the PID bits match the bits
  * specified by the filter.
  *
  * Example: If ID 0x24 is received (PID 0x64), the filter mask (0x30) is applied
  * first (0x64 & 0x30 = 0x20) and then the result is compared to the priority
  * compare data filter (0x20). Since the result matches the filter, the header
  * is received and the user callback is called. Otherwise, the SCI awaits
  * the next break field, and the callback is not called.
  *
  * Note: The most significant 2 bits of the PID are the parity bits, and are
  * masked off by the filter mask in this example.
  */
 sci_b_lin_id_filter_setting_t filter =
    {
       .compare_data_mask             = BASIC_FILTER_MASK,
       .priority_compare_data         = BASIC_PRIORITY_FILTER_DATA,
       .secondary_compare_data       = BASIC_SECONDARY_FILTER_DATA,
       .priority_interrupt_bit_select = BASIC_PRIORITY_INTERRUPT_BIT,
       .priority_interrupt_enable    = SCI_B_LIN_PRIORITY_INTERRUPT_BIT_DISABLE,
       .compare_data_select          = SCI_B_LIN_COMPARE_DATA_SELECT_PRIORITY,
```

```
    };
 /* Configure the ID filter */
    err = R_SCI_B_LIN_IdFilterSet(&g_slave_ctrl, &filter);
    assert(FSP_SUCCESS == err);
 /* Wait for a header which matches the filter */
 while (!g_header_rx_complete)
    {
    }
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = g_received_pid,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data response */
    err = R_SCI_B_LIN_Write(&g_slave_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for the response transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_B_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Advanced LIN Slave ID Filtering Example

The following example demonstrates an advanced example of frame ID filtering.

```
#define ADVANCED_FILTER_MASK (0x3E)
#define ADVANCED_PRIORITY_FILTER_DATA (0x20)
#define ADVANCED_SECONDARY_FILTER_DATA (0x08)
#define ADVANCED_PRIORITY_INTERRUPT_BIT (5)
```

```
void r_sci_b_lin_advanced_id_filtering_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_B_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* If the application is using dynamically assigned identifiers, perform
  * any required initial communications to get the identifiers for this slave */
 /* Configure filter to accept the following frame IDs:
  *
  * - Any ID in which bit 5 is set to 1 (0x20-0x3F).
  * - IDs 0x08 and 0x09
  *
  * The priority compare data filter is set to 0x20 (bit 5 set).
  * The priority interrupt bit select function is enabled for bit 5.
  *
  * The secondary compare data filter is set to 0x08. This will allow
  * both 0x08 and 0x09 through the filter, even though bit 5 is not set for
  * those values (0x08 & 0x3E = 0x08 and 0x09 & 0x3E = 0x08). All other
  * identifiers are filtered out.
  */
 sci_b_lin_id_filter_setting_t filter =
    {
       .compare_data_mask             = ADVANCED_FILTER_MASK,
       .priority_compare_data         = ADVANCED_PRIORITY_FILTER_DATA,
       .secondary_compare_data       = ADVANCED_SECONDARY_FILTER_DATA,
       .priority_interrupt_bit_select = ADVANCED_PRIORITY_INTERRUPT_BIT,
       .priority_interrupt_enable    = SCI_B_LIN_PRIORITY_INTERRUPT_BIT_ENABLE,
       .compare_data_select          = SCI_B_LIN_COMPARE_DATA_SELECT_BOTH,
    };
 /* Configure the ID filter */
    err = R_SCI_B_LIN_IdFilterSet(&g_slave_ctrl, &filter);
    assert(FSP_SUCCESS == err);
 /* Wait for a header which matches the filter */
 while (!g_header_rx_complete)
```

```
    {
    }
lin_transfer_params_t write_params =
    {
        .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
        .id            = g_received_pid,
        .num_bytes     = TRANSFER_LENGTH,
        .p_data        = g_tx_buf
    };
/* Send the LIN data response */
    err = R_SCI_B_LIN_Write(&g_slave_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
/* Wait for the response transmission to complete */
while (!g_data_tx_complete)
    {
    }
/* Close the driver */
    err = R_SCI_B_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | sci_b_lin_timer_setting_t |
| struct | sci_b_lin_baud_setting_t |
| struct | sci_b_lin_id_filter_setting_t |
| struct | sci_b_lin_baud_params_t |
| struct | sci_b_lin_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | sci_b_lin_priority_interrupt_bit_t |
| enum | sci_b_lin_compare_data_select_t |
| enum | sci_b_lin_clock_source_t |

| | |
|---|---|
| enum | sci_b_lin_break_delimiter_bits_t |
| enum | sci_b_lin_timer_divider_t |
| enum | sci_b_lin_synchronization_t |
| enum | sci_b_lin_noise_cancellation_t |
| enum | sci_b_lin_noise_cancellation_clock_t |
| enum | e_sci_b_lin_bus_conflict_detection_t |
| enum | sci_b_lin_bus_conflict_clock_t |
| enum | sci_b_lin_base_clock_t |

## Data Structure Documentation

### ◆ sci_b_lin_timer_setting_t

| struct sci_b_lin_timer_setting_t | | |
|---|---|---|
| Register settings for configuring the LIN break field timer | | |
| Data Fields | | |
| uint8_t | __pad0__: 6 | |
| uint8_t | tcss: 2 | LIN timer count clock source selection. |
| uint16_t | bflw | Break field length setting (break field length is (bflw + 1) * clock of the timer) |

### ◆ sci_b_lin_baud_setting_t

| struct sci_b_lin_baud_setting_t | | |
|---|---|---|
| Register settings for achieving a desired baud rate. | | |
| Data Fields | | |
| union sci_b_lin_baud_setting_t | __unnamed__ | |
| sci_b_lin_timer_setting_t | timer_setting | Break field timer settings associated with this baud rate. |

### ◆ sci_b_lin_id_filter_setting_t

| struct sci_b_lin_id_filter_setting_t | | |
|---|---|---|
| Parameters for configuring the ID filter settings | | |
| Data Fields | | |
| uint8_t | compare_data_mask | Bit mask applied before comparing the received PID to |

| | | |
|---|---|---|
| | | the compare data. Selects which bits of the selected compare data must match. |
| uint8_t | priority_compare_data | Priority/primary compare data |
| uint8_t | secondary_compare_data | Secondary compare data |
| uint8_t | priority_interrupt_bit_select: 3 | Specify ONE of bits 0 to 7 of Control Field 1 as the priority interrupt bit. 0 is bit 0 of the PID, 1 is bit 1 of the PID, and so on. |
| uint8_t | priority_interrupt_enable: 1 | Set to 1 to enable the priority interrupt bit filter specified by priority_interrupt_bit_select, 0 to disable. When this bit is 1, regardless of the compare_data_select setting value, the bit specified by priority_interrupt_bit_select is compared with the priority/primary comparison data for Control Field 1 (priority_compare_data). |
| uint8_t | compare_data_select: 2 | Select the compare data for Control Field 1 (priority, secondary, or both). See sci_b_lin_compare_data_select_t |
| uint8_t | __pad0__: 2 | |

### ◆ sci_b_lin_baud_params_t

| struct sci_b_lin_baud_params_t | | |
|---|---|---|
| Parameters for baud and timer setting calculation | | |
| Data Fields | | |
| uint32_t | baudrate | Desired baudrate. |
| sci_b_lin_clock_source_t | clock_source | Peripheral clock source. |
| sci_b_lin_bus_conflict_clock_t | bus_conflict_clock | DEPRECATED, not used to set baudrate. |
| uint16_t | break_bits | Master mode: Number of break field bits to transmit. Slave mode: Number of break field threshold bits. |

### ◆ sci_b_lin_extended_cfg_t

| struct sci_b_lin_extended_cfg_t |
|---|
| SCI B LIN extended configuration |

| Data Fields | | |
|---|---|---|
| union sci_b_lin_extended_cfg_t | __unnamed__ | |
| sci_b_lin_baud_setting_t | baud_setting | Register settings for a desired baud rate. |
| sci_b_lin_id_filter_setting_t | filter_setting | ID filter setting. |
| IRQn_Type | bfd_irq | Break field detect IRQ number. |
| IRQn_Type | aed_irq | Active edge detect IRQ number. |
| IRQn_Type | rxi_irq | Receive interrupt IRQ number. |
| IRQn_Type | txi_irq | Transmit interrupt IRQ number. |
| IRQn_Type | tei_irq | Transmit end interrupt IRQ number. |
| IRQn_Type | eri_irq | Error interrupt IRQ number. |
| uint8_t | bfd_ipl | Break field detect interrupt priority. |
| uint8_t | aed_ipl | Active edge detect interrupt priority. |
| uint8_t | rxi_ipl | Receive interrupt priority. |
| uint8_t | txi_ipl | Transmit interrupt priority. |
| uint8_t | tei_ipl | Transmit end interrupt priority. |
| uint8_t | eri_ipl | Error interrupt priority. |
| uint16_t | break_bits | Master mode: Number of break field bits to transmit. Slave mode: Number of break field threshold bits. |

## Enumeration Type Documentation

### ◆ sci_b_lin_priority_interrupt_bit_t

| enum sci_b_lin_priority_interrupt_bit_t | |
|---|---|
| Priority interrupt bit options for ID filtering. | |
| Enumerator | |
| SCI_B_LIN_PRIORITY_INTERRUPT_BIT_DISABLE | Disable the priority interrupt bit. |
| SCI_B_LIN_PRIORITY_INTERRUPT_BIT_ENABLE | Enable the priority interrupt bit. |

### ◆ sci_b_lin_compare_data_select_t

| enum sci_b_lin_compare_data_select_t | |
|---|---|
| Compare Data Select options for ID filtering. | |
| Enumerator | |
| SCI_B_LIN_COMPARE_DATA_SELECT_PRIORITY | Select the priority/primary compare data filter as the compare data. |
| SCI_B_LIN_COMPARE_DATA_SELECT_SECONDARY | Select the secondary compare data filter as the compare data. |
| SCI_B_LIN_COMPARE_DATA_SELECT_BOTH | Select both the priority/primary compare data filter and the secondary compare data filter as the compare data. The priority filter will be checked first. |

### ◆ sci_b_lin_clock_source_t

| enum sci_b_lin_clock_source_t | |
|---|---|
| Source clock selection options for SCI. | |
| Enumerator | |
| SCI_B_LIN_CLOCK_SOURCE_SCISPICLK | SCISPI clock source. |
| SCI_B_LIN_CLOCK_SOURCE_SCICLK | SCI clock source. |
| SCI_B_LIN_CLOCK_SOURCE_PCLK | PCLK source. |

### ◆ sci_b_lin_break_delimiter_bits_t

| enum sci_b_lin_break_delimiter_bits_t | |
|---|---|
| Break field delimiter configuration. | |
| Enumerator | |
| SCI_B_LIN_BREAK_DELIMITER_BITS_1 | 1-bit recessive break field delimiter |
| SCI_B_LIN_BREAK_DELIMITER_BITS_2 | 2-bit recessive break field delimiter |

### ◆ sci_b_lin_timer_divider_t

| enum sci_b_lin_timer_divider_t | |
|---|---|
| LIN timer configuration, used for break field generation and detection. | |
| Enumerator | |
| SCI_B_LIN_TIMER_DIV_4 | LIN timer frequency is TCLK/4. |
| SCI_B_LIN_TIMER_DIV_16 | LIN timer frequency is TCLK/16. |
| SCI_B_LIN_TIMER_DIV_64 | LIN timer frequency is TCLK/64. |

### ◆ sci_b_lin_synchronization_t

| enum sci_b_lin_synchronization_t | |
|---|---|
| Auto synchronization setting. | |
| Enumerator | |
| SCI_B_LIN_AUTO_SYNCHRONIZATION_DISABLE | Disable auto synchronization during sync byte reception. |
| SCI_B_LIN_AUTO_SYNCHRONIZATION_ENABLE | Enable auto synchronization during sync byte reception. |

### ◆ sci_b_lin_noise_cancellation_t

| enum sci_b_lin_noise_cancellation_t | |
|---|---|
| Noise cancellation configuration. | |
| Enumerator | |
| SCI_B_LIN_NOISE_CANCELLATION_DISABLE | Disable noise cancellation. |
| SCI_B_LIN_NOISE_CANCELLATION_ENABLE | Enable noise cancellation. |

### ◆ sci_b_lin_noise_cancellation_clock_t

| enum sci_b_lin_noise_cancellation_clock_t | |
|---|---|
| Noise cancellation clock configuration. | |
| Enumerator | |
| SCI_B_LIN_NOISE_CANCELLATION_CLOCK_BASE_CLOCK_DIV_1 | Noise filter clock is the base clock/1. |
| SCI_B_LIN_NOISE_CANCELLATION_CLOCK_BAUDRATE_GENERATOR_CLOCK_DIV_1 | Noise filter clock is on-chip baudrate generator source clock/1. |
| SCI_B_LIN_NOISE_CANCELLATION_CLOCK_BAUDRATE_GENERATOR_CLOCK_DIV_2 | Noise filter clock is on-chip baudrate generator source clock/2. |
| SCI_B_LIN_NOISE_CANCELLATION_CLOCK_BAUDRATE_GENERATOR_CLOCK_DIV_4 | Noise filter clock is on-chip baudrate generator source clock/3. |
| SCI_B_LIN_NOISE_CANCELLATION_CLOCK_BAUDRATE_GENERATOR_CLOCK_DIV_8 | Noise filter clock is on-chip baudrate generator source clock/4. |

### ◆ e_sci_b_lin_bus_conflict_detection_t

| enum e_sci_b_lin_bus_conflict_detection_t | |
|---|---|
| Bus conflict detection enable. | |
| Enumerator | |
| SCI_B_LIN_BUS_CONFLICT_DETECTION_DISABLE | Disable bus conflict detection. |
| SCI_B_LIN_BUS_CONFLICT_DETECTION_ENABLE | Enable bus conflict detection. |

### ◆ sci_b_lin_bus_conflict_clock_t

| enum sci_b_lin_bus_conflict_clock_t | |
|---|---|
| Bus conflict detection clock selection. Base clock: 1/16 period of 1 bit period when CCR2.ABCS = 0, 1/8 period of 1 bit period when CCR2.ABCS = 1 | |
| Enumerator | |
| SCI_B_LIN_BUS_CONFLICT_DETECTION_BASE_CLOCK_DIV_1 | Bus conflict detection clock is base clock. |
| SCI_B_LIN_BUS_CONFLICT_DETECTION_BASE_CLOCK_DIV_2 | Bus conflict detection clock is base clock/2. |
| SCI_B_LIN_BUS_CONFLICT_DETECTION_BASE_CLOCK_DIV_4 | Bus conflict detection clock is base clock/4. Setting prohibited when CCR2.ABCS = 1. |

### ◆ sci_b_lin_base_clock_t

| enum sci_b_lin_base_clock_t | |
|---|---|
| Asynchronous Mode Base Clock Selection | |
| Enumerator | |
| SCI_B_LIN_BASE_CLOCK_16_CYCLES_PER_BIT | Select 16 base clock cycles for 1-bit period. |
| SCI_B_LIN_BASE_CLOCK_8_CYCLES_PER_BIT | Select 8 base clock cycles for 1-bit period. |
| SCI_B_LIN_BASE_CLOCK_AUTO_CYCLES_PER_BIT | Automatic software selection of the base clock cycles (16 or 8) for 1-bit period. |

## Function Documentation

### ◆ R_SCI_B_LIN_Open()

fsp_err_t R_SCI_B_LIN_Open ( lin_ctrl_t *const *p_api_ctrl*, lin_cfg_t const *const *p_cfg* )

Configures the LIN driver channel based on the input configuration.

Implements lin_api_t::open.

Example:

```
/* Open the LIN instance with initial configuration. */
fsp_err_t err = R_SCI_B_LIN_Open(&g_master_ctrl, &g_master_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or configuration structure is NULL. |
| FSP_ERR_INVALID_CHANNEL | The requested channel does not exist on this MCU or the channel does not support Simple LIN mode. |
| FSP_ERR_INVALID_ARGUMENT | Break field length setting or timer divisor is invalid. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_MODE | Setting not supported for selected mode |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_Write()

fsp_err_t R_SCI_B_LIN_Write ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking transmission of a LIN frame.

In master mode, the LIN header (break, sync and protected identifier) is sent before the LIN data.

- To transmit the LIN header only, set p_transfer_params->p_data to NULL.
- To transmit both header and data, set p_transfer_params->p_data to the TX buffer to transmit.

The LIN header is not transmitted in slave mode. Set p_transfer_params->p_data to the TX buffer to transmit.

On successful header-only transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_HEADER_COMPLETE. On successful data transmission or header+data transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_DATA_COMPLETE.

Implements lin_api_t::write.

Example:

```
lin_transfer_params_t write_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = FRAME_ID,
      .num_bytes     = TRANSFER_LENGTH,
      .p_data        = g_tx_buf
   };
/* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
   err = R_SCI_B_LIN_Write(&g_master_ctrl, &write_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transmission started successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | ID out of range (unprotected ID must be less than 64). Only applies in master mode. In slave mode, p_transfer_params->id is ignored. |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SCI_B_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_Read()

fsp_err_t R_SCI_B_LIN_Read ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking data reception to receive user specified number of bytes into destination buffer pointer.

The checksum type specifies the checksum type used for validation. If a non-standard algorithm is used, or the application prefers to validate the checksum outside the driver, or the application prefers to skip checksum validation, specify lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE. If checksum validation is skipped, the lin_checksum_type_t::LIN_EVENT_ERR_INVALID_CHECKSUM event is not possible. When lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE is used, the number of bytes specified in the receive buffer length will be received (the driver will not expect to receive an additional 1 checksum byte), so if a non-standard checksum is used, sufficient space must be allocated in the write buffer and accounted for in the provided length.

On successful data reception completion, the callback is called with event lin_event_t::LIN_EVENT_RX_DATA_COMPLETE.

Implements lin_api_t::read.

Example:

```
/* Begin reception of the data */
   err = R_SCI_B_LIN_Read(&g_master_ctrl, &read_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception started successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, transfer parameters, or tx/rx buffer is NULL, or 0 bytes length provided |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SCI_B_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |
| FSP_ERR_INVALID_CALL | Data reception is not possible because a header frame has not been received yet (slave mode only). |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_CommunicationAbort()

fsp_err_t R_SCI_B_LIN_CommunicationAbort ( lin_ctrl_t *const *p_api_ctrl*)

Cancels in progress data read or write, or header read or write. Break field reception cannot be cancelled. For slave nodes, reception of a new header is still enabled after a call to this function.

Implements lin_api_t::communicationAbort.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transfer aborted successfully or no transfer was in progress. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_CallbackSet()

fsp_err_t R_SCI_B_LIN_CallbackSet ( lin_ctrl_t *const *p_api_ctrl*, void(*)(lin_callback_args_t *) *p_callback*, void *const *p_context*, lin_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure.

Implements lin_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or callback is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_BaudCalculate()

fsp_err_t R_SCI_B_LIN_BaudCalculate ( sci_b_lin_baud_params_t const *const *p_baud_params*, sci_b_lin_baud_setting_t *const *p_baud_setting* )

Calculates baud rate and LIN timer (BFLW and TCSS) register settings. This function evaluates and determines the most accurate settings for the baud rate and timer related registers.

The LIN timer setting is used for break field transmission/detection. Because the timer setting is specified in terms of bits, and the duration of a bit varies depending on baud rate, the baud rate register settings and timer register settings are related. The smallest possible LIN timer divider which can achieve the desired break field bits setting at the configured baud rate is selected to provide the highest measurement accuracy.

The baud rate cannot be updated at runtime with this function. This function is provided to ease configuration of the initial baud settings.

***Note***

*The parameter p_baud_setting->bus_conflict_clock is deprecated and is not used to determine the baud rate settings. See Usage Notes for a description of how to select the bus conflict clock divider.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_baud_params | Parameters required to calculate the baud rate |
| [out] | p_baud_setting | If calculation succeeds, contains computed values to achieve requested baud rate. If calculation fails, the input structure is not modified. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Register settings updated in provided p_baud_setting |
| FSP_ERR_ASSERTION | p_baud_setting was NULL |
| FSP_ERR_INVALID_ARGUMENT | Cannot achieve combination of break field bits and baudrate with provided settings or p_baud_params->baudrate was 0 or p_baud_params->break_bits was 0 |

### ◆ R_SCI_B_LIN_IdFilterSet()

fsp_err_t R_SCI_B_LIN_IdFilterSet ( lin_ctrl_t *const *p_api_ctrl*, sci_b_lin_id_filter_setting_t const *const *p_config* )

Set the the ID filter settings for filtering control field 1 (PID byte).

NOTE: Setting the ID filter will abort any in-progress LIN header reception, as the ID filter settings cannot be changed during reception of the header. The next header will be received with the new settings.

```
/* Configure the ID filter */
   err = R_SCI_B_LIN_IdFilterSet(&g_slave_ctrl, &filter);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the LIN control block. |
| [in] | p_config | The ID filter settings to apply |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID filter updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or p_config is NULL. |
| FSP_ERR_INVALID_MODE | Function called by master node (not supported for master nodes) |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_IdFilterGet()

fsp_err_t R_SCI_B_LIN_IdFilterGet ( lin_ctrl_t *const *p_api_ctrl*, sci_b_lin_id_filter_setting_t *const *p_config* )

Returns the currently configured ID filter settings.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the LIN control block. |
| [out] | p_config | The current ID filter settings |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID filter updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or p_config is NULL. |
| FSP_ERR_INVALID_MODE | Function called by master node (not supported for master nodes) |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_WakeupSend()

fsp_err_t R_SCI_B_LIN_WakeupSend ( lin_ctrl_t *const *p_api_ctrl*)

Send wakeup is not supported on the SCI_B LIN driver.

Implements lin_api_t::wakeupSend.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_B_LIN_SleepEnter()

fsp_err_t R_SCI_B_LIN_SleepEnter ( lin_ctrl_t *const *p_api_ctrl*)

Sleep is not supported on the SCI_B LIN driver.

Implements lin_api_t::sleepEnter.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_B_LIN_SleepExit()

fsp_err_t R_SCI_B_LIN_SleepExit ( lin_ctrl_t *const *p_api_ctrl*)

Sleep is not supported on the SCI_B LIN driver.

Implements lin_api_t::sleepExit.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_B_LIN_Close()

fsp_err_t R_SCI_B_LIN_Close ( lin_ctrl_t *const *p_api_ctrl*)

Closes the LIN driver.

Implements lin_api_t::close.

Example:

```
/* Close the driver */
   err = R_SCI_B_LIN_Close(&g_master_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_B_LIN_InformationFrameRead()

| fsp_err_t R_SCI_B_LIN_InformationFrameRead ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* ) |
|---|
| [DEPRECATED] Use R_SCI_B_LIN_Read.<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

### ◆ R_SCI_B_LIN_StartFrameWrite()

| fsp_err_t R_SCI_B_LIN_StartFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, uint8_t const *id* ) |
|---|
| [DEPRECATED] Use R_SCI_B_LIN_Write<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

### ◆ R_SCI_B_LIN_InformationFrameWrite()

| fsp_err_t R_SCI_B_LIN_InformationFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* ) |
|---|
| [DEPRECATED] Use R_SCI_B_LIN_Write<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

## 5.2.6.19 LIN (r_sci_lin)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_LIN_Open (lin_ctrl_t *const p_api_ctrl, lin_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_LIN_Write (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_LIN_Read (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_LIN_CommunicationAbort (lin_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | R_SCI_LIN_CallbackSet (lin_ctrl_t *const p_api_ctrl, void(*p_callback)(lin_callback_args_t *), void *const p_context, lin_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_LIN_BaudCalculate (sci_lin_baud_params_t const *const p_baud_params, sci_lin_baud_setting_t *const p_baud_setting) |
| fsp_err_t | R_SCI_LIN_IdFilterSet (lin_ctrl_t *const p_api_ctrl, sci_lin_id_filter_setting_t const *const p_config) |
| fsp_err_t | R_SCI_LIN_IdFilterGet (lin_ctrl_t *const p_api_ctrl, sci_lin_id_filter_setting_t *const p_config) |
| fsp_err_t | R_SCI_LIN_Close (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_LIN_WakeupSend (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_LIN_SleepEnter (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_LIN_SleepExit (lin_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_LIN_InformationFrameRead (lin_ctrl_t *const p_api_ctrl, lin_transfer_params_t *const p_transfer_params) |
| fsp_err_t | R_SCI_LIN_StartFrameWrite (lin_ctrl_t *const p_api_ctrl, uint8_t const id) |
| fsp_err_t | R_SCI_LIN_InformationFrameWrite (lin_ctrl_t *const p_api_ctrl, const lin_transfer_params_t *const p_transfer_params) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the LIN Interface.

# Overview

The Simple LIN on SCI HAL module supports Local Interface Network (LIN) transactions in master or slave mode. A callback must be provided, and is invoked when a transmission, reception, or other event has completed or occurred. The callback arguments contain information about the transaction status, bytes transferred and a pointer to the user defined context.

*Note*

*Some MCUs call this feature **Extended Serial Mode**. **Extended Serial Mode** and **Simple Lin** are equivalent terms.*

### Features

- Half-duplex master or slave mode LIN communication
- Interrupt-driven data transmission and reception
- Generation of the sync (0x55) and Protected Identifier fields when in master mode

- Validation of the Protected Identifier field when in slave mode
- ID filtering support in slave mode
- Optional checksum generation/validation using classic or enhanced LIN checksum
- Invoking the user-callback function with an event code (RX/TX complete, error, etc)
- Adjustable break field length
- Adjustable break field delimiter bits length
- Digital filter
- Abort in-progress read/write operations
- Error notifications of parity error in protected identifier, framing error, overrun error, bus collision
- Configuration of all available SCI channels with Simple LIN mode support
- Auto synchronization is supported in slave mode

# Configuration

### Build Time Configurations for r_sci_lin

The following build time configurations are defined in fsp_cfg/r_sci_lin_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Checksum Support | • Enabled<br>• Disabled | Enabled | When set to 'Disabled', code for checksum generation and validation is excluded from the build. This setting is applied globally to the project. Disable only when checksum generation and validation is not required for any LIN instance. |
| Auto Synchronization Support | • Enabled<br>• Disabled | Disabled | When set to 'Disabled', code for auto synchronization is excluded from the build. This setting is applied globally to the project. Enable when at least one LIN slave instance is using auto synchronization. |
| Bus Conflict Detection Support | • Enabled<br>• Disabled | Disabled | When set to 'Disabled', code for bus conflict detection is excluded from the build. This setting is applied globally to the project. Enable when at least |

| | | | |
|---|---|---|---|
| | | | one LIN instance is using bus conflict detection. |

### Configurations for Connectivity > LIN (r_sci_lin)

This module can be added to the Stacks tab via New Stack > Connectivity > LIN (r_sci_lin). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_lin0 | Module name. |
| SCI Channel | MCU Specific Options | | Select the Serial Communications Interface channel. |
| Mode | • Master<br>• Slave | Master | Select the LIN operating mode (master or slave). |
| Extra | | | |
| Digital Filter Clock Select | • Disabled<br>• SCI Base Clock<br>• PCLK/8<br>• PCLK/16<br>• PCLK/32<br>• PCLK/64<br>• PCLK/128 | Disabled | Enable the digital filter on RXDXn pin. The digital filter block in SCI consists of three-stage flipflop circuits. |
| Bus Conflict Clock Divider | • 1<br>• 2<br>• 4 | 2 | Select the base clock divider for the sampling clock of the bus conflict detection circuit. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 19200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated sci_lin_extended_cfg_t structure. |

| | | | |
|---|---|---|---|
| Base Clock Cycles Per Bit Period | • 8<br>• 16 | 16 | Select number of SCI base clock cycles per single bit period at the configured baud rate. When 16 base clock cycles is selected, the period of the SCI base clock is 1/16 of a single bit period. When 8 base clock cycles is selected, the period of the SCI base clock is 1/8 of a single bit period. |
| Framing | | | |
| Framing > ID Filter (Slave Mode) | | | |
| Compare Data Mask | Value must be an integer between 0 and 255. | 0 | Select the bit mask to be applied before comparing the received PID to the selected compare data. This setting specifies which bits of the selected compare data must match. Set to 0 to disable the filter (allow all frame identifiers). |
| Primary Compare Data | Value must be an integer between 0 and 255. | 0 | Select the primary compare filter data. |
| Secondary Compare Data | Value must be an integer between 0 and 255. | 0 | Select the secondary compare filter data. |
| Compare Data Select | • Primary<br>• Secondary<br>• Both | Primary | Select the compare data to use. If 'both' is selected, the primary compare data is checked before the secondary compare data. |
| Priority Interrupt Bit Enable | • Enabled<br>• Disabled | Disabled | Select whether to enable the Priority Interrupt Bit filter. When enabled, regardless of the Filter Data Select setting value, the bit specified by the Priority Interrupt Bit setting is compared with the corresponding bit in the Primary |

| | | | |
|---|---|---|---|
| | | | Compare Data Filter and if it matches, the identifier is allowed through the filter. |
| Priority Interrupt Bit | Value must be an integer between bit 0 (LSb) and bit 7 (MSb). | 0 | Select the priority interrupt bit (0-7) of the received PID to compare with the corresponding bit in the Primary Compare Data Filter. |
| Break Field Bits/Break Detection Threshold (bits) | Value must be an integer. | 13 | For master break field transmission, this configures the dominant period of the break field (in bits). For slave break field reception, this configures the break detection threshold in bits. Must be 13 bits or greater for master mode and 11 bits or greater for slave mode. |
| Break Delimiter Bits | Value must be an integer between 1 and 255. | 1 | This configures the period of the break delimiter (in bits).The break delimiter shall be at least one nominal bit time long. |
| LIN Timer Divider | • 1<br>• 2<br>• 4<br>• 8<br>• 16<br>• 32<br>• 64<br>• 128 | 1 | Set the LIN timer divider. The LIN timer is used for break field transmission and detection. Higher dividers make possible transmission/detection of more break field bits, but are less precise. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | sci_lin0_callback | A user callback function must be provided. It will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Data Empty | MCU Specific Options | | Select the transmit |

| | | |
|---|---|---|
| Interrupt Priority | | interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | Select the transmit end interrupt priority. |
| Receive Error Interrupt Priority | MCU Specific Options | Select the receive error interrupt priority. |
| Break Field Low Width Interrupt Priority (SCIX0) | MCU Specific Options | Select the break field low width interrupt priority. |
| Control Field 0/1 Match Interrupt Priority (SCIX1) | MCU Specific Options | Select the Control field 0,1 and Priority bit detection interrupt priority. |
| Bus Collision Detection Interrupt Priority (SCIX2) | MCU Specific Options | Select the bus collision detection interrupt priority, or set to 'Disabled' to disable bus collision detection. |
| Valid Edge Detection Interrupt Priority (SCIX3) | MCU Specific Options | Select the valid edge detection interrupt priority to enable slave auto synchronization during the LIN sync byte, or set to 'Disabled' to disable auto synchronization. |

### Clock Configuration

Peripheral clock for SCI LIN is PCLKA.

### Pin Configuration

This module uses TXD and RXD to communicate to external devices. Connect TXD and RXD to an on-board or external LIN transceiver for LIN bus communication.

An external pull up resistor is required on TXD in both master and slave mode.

### Interrupt Configuration

- Receive data full (RXI), Transmit data empty (TXI), Transmit end (TEI), Error event (ERI), Break Field low width (SCIX0) interrupts for the selected channel used must be enabled in the properties of the selected device for both master and slave mode operation.
- Control field 0/1 match, priority interrupt bit detection (SCIX1) interrupt is only used in slave mode.
- Configuring the bus collision detection (SCIX2) interrupt is required to enable Bus Conflict Detection.
- Configuring the valid edge detection (SCIX3) interrupt is required to enable Auto Synchronization (slave mode only).

### Break Field Length/Detection Threshold Configuration

- When in master mode, this setting configures the low (dominant) time of the LIN break pattern.
- When in slave mode, this setting configures the break field detection threshold (number of bits that must be received to detect a break field).
- When configuring this setting in slave mode, take care to set the detection threshold less than the master break field length.
- The standard settings are 13 bits for master nodes and 11 bits for slave nodes, but alternate settings are supported.

**Bus Conflict Detection Configuration**

When the Bus Collision Detection Interrupt (SCIX2) is enabled, the TXDn pin output and the RXDn pin input are sampled by the selected bus conflict detection clock. When a mismatch occurs three times in a row, an SCIX2 interrupt is generated and transmission is stopped. This function only works during transmission.

Consideration of the specific application's LIN bus characteristics should be taken into account when enabling this function. If the bus collision sampling clock is too fast, the expected propagation delay through the transceiver may be erroneously detected as a bus collision (false positive). If the bus collision sampling clock is too slow, collisions may be missed (false negative).

A LIN transceiver is required to detect bus collisions. The propagation delay through the transceiver varies depending on individual transceiver design.

When Bus Conflict Detection is enabled, set the Bus Conflict Clock Divider to divide down the clock at least slow enough to avoid false positive collision detections caused by the worst case transceiver propagation delay. The Baud|Base Clock Cycles Per Bit Period setting can be used to adjust the SCI Base Clock frequency if needed.

***Example:***

The SCI base clock frequency can be calculated by the following formula:

sci_base_clock_frequency = 1 / [ (1/baud_rate)/cycles_per_bit_period ]

The following relationship must be satisfied to avoid false positives:

2 *[ 1e6 / (sci_base_clock_frequency/bus_conflict_clock_divider) ] > max_propagation_delay_us

Suppose:

- Transceiver TXD->LIN BUS max delay = 6us
- Transceiver LIN BUS->RXD max delay = 6us
- Transceiver max_propagation_delay_us: 6 + 6 = 12us
- baud_rate = 19200
- cycles_per_bit_period = 16

Then:

- 1 bit time @ 19200bps = 1e6/19200 = 52us
- sci_base_clock_frequency = 1/[ (1/19200)/16 ] = 307200 Hz

Bus Conflict Divider Selection Process:

- Divider of 1:
  - 2 * [ 1e6 / (307200/1) ] = 6.51 us (not > 12us, may cause false positives)
- Divider of 2:
  - 2 * [ 1e6 / (307200/2) ] = 13.02 us (is > 12us, can be used)
- Divider of 4:
  - 2 * [ 1e6 / (307200/4) ] = 26.04 us (is > 12us, can be used, but will be less accurate at detecting real collisions than divider of 2 due to slower bus collision clock)

If no possible divider is found with 16 cycles, Baud|Base Clock Cycles Per Bit Period can be reduced to 8, which will reduce the SCI Base Clock by half compared to 16.

### ID Filter Configuration

The SCI performs optional hardware PID filtering in slave mode. When enabled, only headers with PIDs that pass through the filter are received and passed to the user callback. Other headers are ignored.

An initial configuration of the filter settings can be applied in the properties view by selecting Framing -> ID Filter and choosing the desired settings. See sci_lin_id_filter_setting_t for the available filter settings, as well as the examples in this document. By default, filtering is disabled and all PIDs can be received.

The filter can be updated at runtime by using using R_SCI_LIN_IdFilterSet. The current filter settings can be viewed with R_SCI_LIN_IdFilterGet.

# Usage Notes

### Bit Rate Calculation

For more details on how the baudrate is set refer to Table "Relationship between N setting in BRR and bit rate B" in the SCI section of the relevant hardware manual.

### LIN Transmission and Reception

SCI data reception is disabled until a break field is detected (slave mode) or a call to R_SCI_LIN_Read() is made (master and slave mode).

### Slave Mode Transmission and Reception

The header is detected in slave mode only. Header reception is interrupt driven, and is enabled after a successful call to R_SCI_LIN_Open() in slave mode.

When the LIN break field is detected by the slave node, header reception begins without action from the application. When header reception completes, the user callback is called.

If the slave node needs to receive the data frame, the slave node must call R_SCI_LIN_Read() before the first data is received in order to receive the data. The user callback is called when reception is completed. It is permitted to call R_SCI_LIN_Read() from the callback context.

If the slave node needs to publish a response to the header, the slave node must call R_SCI_LIN_Write() and provide the response data and the PID of the received header in the lin_transfer_params_t. The user callback is called when transmission is complete.

**Master Mode Transmission and Reception**

The header is not detected in master mode. Data frame reception is enabled only after a call to R_SCI_LIN_Read().

The header is transmitted by calling R_SCI_LIN_Write(). The user callback is called when transmission is complete.

If the master node needs to receive the data response, call R_SCI_LIN_Read() after header transmission has completed and before the first data is received. It is permitted to call R_SCI_LIN_Read() from the callback context.

If the master node needs to publish a response to its own header, the master node must call R_SAU_LIN_Write() and provide a pointer to the response data. The response will be published following the header. The user callback is called when the full frame transmission is complete.

**Timeouts/Errors**

The application is responsible for managing timeouts in case R_SCI_LIN_Read() is called, but the data is not received in the expected time period. If a timeout occurs, R_SCI_LIN_CommunicationAbort() may be called to cancel a pending read.

When an error occurs, reception is stopped and the user callback is called with the relevant error code.

**Checksum Generation and Validation**

- The checksum calculation is performed in software for both checksum generation and validation.
- Checksum generation and validation are both optional.
- The LIN checksum may be generated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when writing data.
- The LIN checksum may be validated by passing lin_transfer_params_t::LIN_CHECKSUM_TYPE_CLASSIC or lin_transfer_params_t::LIN_CHECKSUM_TYPE_ENHANCED when reading data.
- To skip driver checksum generation/validation, pass lin_transfer_params_t::LIN_CHECKSUM_TYPE_NONE to the read/write functions. Instead the checksum can be included as part of the data buffer, or omitted entirely depending on the application's needs.
- The maximum number of data bytes that can be transmitted per frame is 255, including the checksum byte. Thus, when checksum generation/validation is requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 254. When checksum generation/validation is not requested in the transfer parameters, the maximum number of bytes that can be specified in transfer_params.num_bytes is 255.

**Auto Synchronization**

Auto synchronization is optional. When it is disabled, the baud rate and break field detection threshold set in open are never updated.

This feature works in slave mode only.

- Auto synchronization is not the same as automatic baud rate detection. Auto synchronization is used for correcting small clock drift/difference in baud rate between nodes.
- Auto synchronization performs small automatic baud rate adjustments to synchronize with the master's clock by measuring the effective edges of the input signal from the RXDn pin during reception of the LIN sync byte.
- The sync byte is measured and the baud setting and break field detection threshold are re-adjusted for every LIN sync byte received.
- If reception of the sync byte fails, or the required register settings detected would not be possible in the hardware, the baud/timer settings are not updated, and auto synchronization will be re-attempted on the next received start frame.

### Limitations

- The SCI provides an interface to the LIN Physical layer and does not implement the LIN Transport Layer or LIN Application Layer. The application is responsible for handling the frame scheduling table and monitoring for Transport Layer timeouts.
- Selectable timing for the sampling data is not supported.
- A new header cannot be detected during active reception of control field 0, control field 1, or frame data.
- Two or more headers that are not followed by data (from any node) cannot be received consecutively unless R_SCI_LIN_CommunicationAbort is called after receiving LIN_EVENT_HEADER_RX_COMPLETE to ignore the frame. If all headers are followed by data, no action is required to ignore a received header.

## Examples

### Basic LIN Master Read Example

The following demonstrates a basic example of frame reception for a LIN master.

```
void r_sci_lin_basic_master_read_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Send the header only */
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = 0,
       .p_data        = NULL          /* Set to NULL to send header only */
    };
```

```
    err = R_SCI_LIN_Write(&g_master_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for header transmission to complete before starting a read of the response
*/
 while (!g_header_tx_complete)
    {
    }
 lin_transfer_params_t read_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_rx_buf
    };
 /* Begin reception of the data */
    err = R_SCI_LIN_Read(&g_master_ctrl, &read_params);
    assert(FSP_SUCCESS == err);
 /* Wait for data reception to complete. The application is responsible for
  * timing out if data is not received within the expected time interval*/
 while (!g_data_rx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
void master_callback (lin_callback_args_t * p_args)
{
 /* Handle the LIN event */
 switch (p_args->event)
    {
 case LIN_EVENT_RX_DATA_COMPLETE:
      {
            g_data_rx_complete = 1;
```

```
break;
     }
case LIN_EVENT_TX_HEADER_COMPLETE:
     {
           g_header_tx_complete = 1;
break;
     }
case LIN_EVENT_TX_DATA_COMPLETE:
     {
           g_data_tx_complete = 1;
break;
     }
case LIN_EVENT_ERR_INVALID_CHECKSUM:
case LIN_EVENT_ERR_BUS_COLLISION_DETECTED:
case LIN_EVENT_ERR_FRAMING:
case LIN_EVENT_ERR_COUNTER_OVERFLOW:
case LIN_EVENT_ERR_OVERRUN:
case LIN_EVENT_ERR_PARITY:
default:
     {
/* Handle error */
     }
   }
}
```

### Basic LIN Master Write Example

The following demonstrates a basic example of frame transmission for a LIN master.

```
void r_sci_lin_basic_master_write_example (void)
{
/* Initialize transmit buffer to known data. */
for (uint8_t i = 0; i < TRANSFER_LENGTH; i++)
   {
      g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
```

```
    }
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_LIN_Open(&g_master_ctrl, &g_master_cfg);
    assert(FSP_SUCCESS == err);
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = FRAME_ID,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
    err = R_SCI_LIN_Write(&g_master_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_LIN_Close(&g_master_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Basic LIN Slave Example

The following demonstrates a basic example of header reception and data response for a LIN slave.

```
void r_sci_lin_basic_slave_example (void)
{
 /* Initialize transmit buffer to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_tx_buf[i] = (uint8_t) ('A' + (i % 26));
    }
```

```
/* Open the LIN instance with initial configuration. */
fsp_err_t err = R_SCI_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
   assert(FSP_SUCCESS == err);
/* Wait for the header that this slave publishes the response to. */
for ( ; ; )
   {
if (g_header_rx_complete)
     {
if (g_received_pid == FRAME_ID)
     {
/* Frame ID of interest received */
break;
     }
/* Ignore this header. No action required by slave. */
           g_header_rx_complete = false;
/* Note: Only in case header will not followed any data,
 * R_SCI_LIN_CommunicationAbort should be called to ignore the header. */
           err = R_SCI_LIN_CommunicationAbort(&g_slave_ctrl);
     assert(FSP_SUCCESS == err);
     }
   }
lin_transfer_params_t write_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = g_received_pid,
      .num_bytes     = TRANSFER_LENGTH,
      .p_data        = g_tx_buf
   };
/* Send the LIN data response */
   err = R_SCI_LIN_Write(&g_slave_ctrl, &write_params);
   assert(FSP_SUCCESS == err);
/* Wait for the response transmission to complete */
while (!g_data_tx_complete)
   {
```

```
    }
 /* Close the driver */
    err = R_SCI_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
void slave_callback (lin_callback_args_t * p_args)
{
 /* Handle the LIN event */
 switch (p_args->event)
    {
 case LIN_EVENT_RX_HEADER_COMPLETE:
      {
            g_header_rx_complete = 1;
            g_received_pid       = p_args->pid;
 break;
      }
 case LIN_EVENT_RX_DATA_COMPLETE:
      {
            g_data_rx_complete = 1;
 break;
      }
 case LIN_EVENT_TX_DATA_COMPLETE:
      {
            g_data_tx_complete = 1;
 break;
      }
 default:
      {
 /* Handle error */
      }
    }
}
```

### Basic LIN Slave ID Filtering Example

The following example demonstrates a basic example of frame ID filtering.

```
#define BASIC_FILTER_MASK (0x30)
#define BASIC_PRIORITY_FILTER_DATA (0x20)
#define BASIC_SECONDARY_FILTER_DATA (0x00) // Don't-care value in this example
#define BASIC_PRIORITY_INTERRUPT_BIT (0) // Don't-care value in this example
void r_sci_lin_basic_id_filtering_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* If the application is using dynamically assigned identifiers, perform
  * any required initial communications to get the identifiers for this slave */
 /* Configure filter to accept only frame IDs in the range 0x20-0x2F.
  *
  * The header is received only when the PID bits match the bits
  * specified by the filter.
  *
  * Example: If ID 0x24 is received (PID 0x64), the filter mask (0x30) is applied
  * first (0x64 & 0x30 = 0x20) and then the result is compared to the priority
  * compare data filter (0x20). Since the result matches the filter, the header
  * is received and the user callback is called. Otherwise, the SCI awaits
  * the next break field, and the callback is not called.
  *
  * Note: The most significant 2 bits of the PID are the parity bits, and are
  * masked off by the filter mask in this example.
  */
 sci_lin_id_filter_setting_t filter =
    {
       .compare_data_mask             = BASIC_FILTER_MASK,
       .primary_compare_data          = BASIC_PRIORITY_FILTER_DATA,
       .secondary_compare_data       = BASIC_SECONDARY_FILTER_DATA,
       .priority_interrupt_bit_select = BASIC_PRIORITY_INTERRUPT_BIT,
       .priority_interrupt_enable    = SCI_LIN_PRIORITY_INTERRUPT_BIT_DISABLE,
       .compare_data_select          = SCI_LIN_COMPARE_DATA_SELECT_PRIMARY,
```

```
    };
 /* Configure the ID filter */
    err = R_SCI_LIN_IdFilterSet(&g_slave_ctrl, &filter);
    assert(FSP_SUCCESS == err);
 /* Wait for a header which matches the filter */
 while (!g_header_rx_complete)
    {
    }
 lin_transfer_params_t write_params =
    {
       .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
       .id            = g_received_pid,
       .num_bytes     = TRANSFER_LENGTH,
       .p_data        = g_tx_buf
    };
 /* Send the LIN data response */
    err = R_SCI_LIN_Write(&g_slave_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for the response transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Advanced LIN Slave ID Filtering Example

The following example demonstrates an advanced example of frame ID filtering.

```
#define ADVANCED_FILTER_MASK (0x3E)
#define ADVANCED_PRIORITY_FILTER_DATA (0x20)
#define ADVANCED_SECONDARY_FILTER_DATA (0x08)
#define ADVANCED_PRIORITY_INTERRUPT_BIT (5)
```

```
void r_sci_lin_advanced_id_filtering_example (void)
{
 /* Open the LIN instance with initial configuration. */
 fsp_err_t err = R_SCI_LIN_Open(&g_slave_ctrl, &g_slave_cfg);
    assert(FSP_SUCCESS == err);
 /* If the application is using dynamically assigned identifiers, perform
  * any required initial communications to get the identifiers for this slave */
 /* Configure filter to accept the following frame IDs:
  *
  * - Any ID in which bit 5 is set to 1 (0x20-0x3F).
  * - IDs 0x08 and 0x09
  *
  * The priority compare data filter is set to 0x20 (bit 5 set).
  * The priority interrupt bit select function is enabled for bit 5.
  *
  * The secondary compare data filter is set to 0x08. This will allow
  * both 0x08 and 0x09 through the filter, even though bit 5 is not set for
  * those values (0x08 & 0x3E = 0x08 and 0x09 & 0x3E = 0x08). All other
  * identifiers are filtered out.
  */
 sci_lin_id_filter_setting_t filter =
    {
       .compare_data_mask             = ADVANCED_FILTER_MASK,
       .primary_compare_data          = ADVANCED_PRIORITY_FILTER_DATA,
       .secondary_compare_data       = ADVANCED_SECONDARY_FILTER_DATA,
       .priority_interrupt_bit_select = ADVANCED_PRIORITY_INTERRUPT_BIT,
       .priority_interrupt_enable    = SCI_LIN_PRIORITY_INTERRUPT_BIT_ENABLE,
       .compare_data_select          = SCI_LIN_COMPARE_DATA_SELECT_BOTH,
    };
 /* Configure the ID filter */
    err = R_SCI_LIN_IdFilterSet(&g_slave_ctrl, &filter);
    assert(FSP_SUCCESS == err);
 /* Wait for a header which matches the filter */
 while (!g_header_rx_complete)
```

```
    {
    }
 lin_transfer_params_t write_params =
    {
        .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
        .id            = g_received_pid,
        .num_bytes     = TRANSFER_LENGTH,
        .p_data        = g_tx_buf
    };
 /* Send the LIN data response */
    err = R_SCI_LIN_Write(&g_slave_ctrl, &write_params);
    assert(FSP_SUCCESS == err);
 /* Wait for the response transmission to complete */
 while (!g_data_tx_complete)
    {
    }
 /* Close the driver */
    err = R_SCI_LIN_Close(&g_slave_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | sci_lin_timer_setting_t |
| struct | sci_lin_baud_setting_t |
| struct | sci_lin_id_filter_setting_t |
| struct | sci_lin_baud_params_t |
| struct | sci_lin_extended_cfg_t |
| struct | sci_lin_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sci_lin_priority_interrupt_bit_t |
| enum | sci_lin_compare_data_select_t |

| | |
|---|---|
| enum | sci_lin_timer_divider_t |
| enum | sci_lin_digital_filter_clock_t |
| enum | sci_lin_bus_conflict_clock_t |
| enum | sci_lin_base_clock_t |

## Data Structure Documentation

### ◆ sci_lin_timer_setting_t

| struct sci_lin_timer_setting_t | | |
|---|---|---|
| Register settings for configuring the LIN timer | | |
| Data Fields | | |
| uint8_t | tcss | LIN timer count clock source selection. |
| uint8_t | break_field_tpre | Break field prescaler. |
| uint8_t | break_field_tcnt | Break field count value. |
| uint8_t | delimiter_tpre | Break delimiter prescaler. |
| uint8_t | delimiter_tcnt | Break delimiter count value. |

### ◆ sci_lin_baud_setting_t

| struct sci_lin_baud_setting_t | | |
|---|---|---|
| Register settings for achieving a desired baud rate. | | |
| Data Fields | | |
| sci_lin_timer_setting_t | timer_setting | Break field timer settings associated with this baud rate. |
| union sci_lin_baud_setting_t | __unnamed__ | |
| uint8_t | brr | Bit Rate Register setting. |
| uint8_t | cks: 2 | CKS value to get divisor. |

### ◆ sci_lin_id_filter_setting_t

| struct sci_lin_id_filter_setting_t | | |
|---|---|---|
| Parameters for configuring the ID filter settings | | |
| Data Fields | | |
| uint8_t | compare_data_mask | Bit mask applied before comparing the received PID to the compare data. Selects which bits of the selected compare data must match. |
| | | |

| | | |
|---|---|---|
| uint8_t | primary_compare_data | Primary compare data |
| uint8_t | secondary_compare_data | Secondary compare data |
| uint8_t | priority_interrupt_bit_select: 3 | Specify ONE of bits 0 to 7 of Control Field 1 as the priority interrupt bit. 0 is bit 0 of the PID, 1 is bit 1 of the PID, and so on. |
| uint8_t | priority_interrupt_enable: 1 | Set to 1 to enable the priority interrupt bit filter specified by priority_interrupt_bit_select, 0 to disable. When this bit is 1, regardless of the compare_data_select setting value, the bit specified by priority_interrupt_bit_select is compared with the Primary comparison data for Control Field 1 (primary_compare_data). |
| uint8_t | compare_data_select: 2 | Select the compare data for Control Field 1 (priority, secondary, or both). See sci_lin_compare_data_select_t |
| uint8_t | __pad0__: 2 | |

### ◆ sci_lin_baud_params_t

| struct sci_lin_baud_params_t | | |
|---|---|---|
| Parameters for baud and timer setting calculation | | |
| Data Fields | | |
| uint32_t | baudrate | Desired baudrate. |
| uint16_t | break_bits | Master mode: Number of break field bits to transmit. Slave mode: Number of break field threshold bits. |
| uint8_t | delimiter_bits | Break delimeter length setting (at least 1 nominal bit time long). |
| sci_lin_base_clock_t | base_clock_cycles_per_bit | Number of base clock cycles for 1-bit period. |

### ◆ sci_lin_extended_cfg_t

| struct sci_lin_extended_cfg_t | | |
|---|---|---|
| SCI LIN extended configuration | | |
| Data Fields | | |
| sci_lin_digital_filter_clock_t | digital_filter_clock | Digital filter setting. See |

| | | |
|---|---|---|
| | | sci_lin_digital_filter_clock_t. |
| sci_lin_bus_conflict_clock_t | bus_conflict_clock | Bus conflict detection clock setting. See sci_lin_bus_conflict_clock_t. |
| uint16_t | break_bits | Master mode: Number of break field bits to transmit. Slave mode: Number of break field threshold bits. |
| uint8_t | delimiter_bits | Break delimeter length setting (at least 1 nominal bit time long). |
| sci_lin_baud_setting_t * | p_baud_setting | Register settings for a desired baud rate. |
| sci_lin_id_filter_setting_t | filter_setting | ID filter setting. |
| IRQn_Type | scix0_irq | Break field low width IRQ number. |
| IRQn_Type | scix1_irq | Control field 0-1 match / Priority interrupt bit detect IRQ number. |
| IRQn_Type | scix2_irq | Bus collision detect IRQ number. |
| IRQn_Type | scix3_irq | Valid edge detect IRQ number. |
| IRQn_Type | rxi_irq | Receive interrupt IRQ number. |
| IRQn_Type | txi_irq | Transmit interrupt IRQ number. |
| IRQn_Type | tei_irq | Transmit end interrupt IRQ number. |
| IRQn_Type | eri_irq | Error interrupt IRQ number. |
| uint8_t | scix0_ipl | Break field low width interrupt priority. |
| uint8_t | scix1_ipl | Control field 0-1 match / Priority interrupt bit detect interrupt priority. |
| uint8_t | scix2_ipl | Bus collision detect interrupt priority. |
| uint8_t | scix3_ipl | Valid edge detect interrupt priority. |
| uint8_t | rxi_ipl | Receive interrupt priority. |
| uint8_t | txi_ipl | Transmit interrupt priority. |
| uint8_t | tei_ipl | Transmit end interrupt priority. |
| uint8_t | eri_ipl | Error interrupt priority. |

### ◆ sci_lin_instance_ctrl_t

| struct sci_lin_instance_ctrl_t |
| --- |
| LIN Instance Control Block. DO NOT INITIALIZE. |

## Enumeration Type Documentation

### ◆ sci_lin_priority_interrupt_bit_t

| enum sci_lin_priority_interrupt_bit_t | |
| --- | --- |
| Priority interrupt bit options for ID filtering. | |
| Enumerator | |
| SCI_LIN_PRIORITY_INTERRUPT_BIT_DISABLE | Disable the priority interrupt bit. |
| SCI_LIN_PRIORITY_INTERRUPT_BIT_ENABLE | Enable the priority interrupt bit. |

### ◆ sci_lin_compare_data_select_t

| enum sci_lin_compare_data_select_t | |
| --- | --- |
| Compare Data Select options for ID filtering. | |
| Enumerator | |
| SCI_LIN_COMPARE_DATA_SELECT_PRIORITY | Deprecated. |
| SCI_LIN_COMPARE_DATA_SELECT_PRIMARY | Select the primary compare data filter as the compare data. |
| SCI_LIN_COMPARE_DATA_SELECT_SECONDARY | Select the secondary compare data filter as the compare data. |
| SCI_LIN_COMPARE_DATA_SELECT_BOTH | Select both the priority/primary compare data filter and the secondary compare data filter as the compare data. The priority filter will be checked first. |

### ◆ sci_lin_timer_divider_t

| enum sci_lin_timer_divider_t | |
|---|---|
| LIN timer divider selection. | |
| Enumerator | |
| SCI_LIN_TIMER_DIV_1 | LIN timer frequency is PCLK/1. |
| SCI_LIN_TIMER_DIV_2 | LIN timer frequency is PCLK/2. |
| SCI_LIN_TIMER_DIV_4 | LIN timer frequency is PCLK/4. |
| SCI_LIN_TIMER_DIV_8 | LIN timer frequency is PCLK/8. |
| SCI_LIN_TIMER_DIV_16 | LIN timer frequency is PCLK/16. |
| SCI_LIN_TIMER_DIV_32 | LIN timer frequency is PCLK/32. |
| SCI_LIN_TIMER_DIV_64 | LIN timer frequency is PCLK/64. |
| SCI_LIN_TIMER_DIV_128 | LIN timer frequency is PCLK/128. |

### ◆ sci_lin_digital_filter_clock_t

| enum sci_lin_digital_filter_clock_t | |
|---|---|
| Digital filter configuration. | |
| Enumerator | |
| SCI_LIN_DIGITAL_FILTER_CLOCK_DISABLED | Filter is disabled. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_SCI_BASE_CLK | Filter clock is SCI base clock. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_PCLK_DIV_8 | Filter clock is PCLK/8. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_PCLK_DIV_16 | Filter clock is PCLK/16. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_PCLK_DIV_32 | Filter clock is PCLK/32. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_PCLK_DIV_64 | Filter clock is PCLK/64. |
| SCI_LIN_DIGITAL_FILTER_CLOCK_PCLK_DIV_128 | Filter clock is PCLK/128. |

### ◆ sci_lin_bus_conflict_clock_t

| enum sci_lin_bus_conflict_clock_t | |
|---|---|
| Bus conflict detection clock selection. The period of the SCI base clock is 1/16 of a single bit period when the SCIn.SEMR.ABCS is 0, and 1/8 of a single bit period when the SCIn.SEMR.ABCS is 1. | |
| Enumerator | |
| SCI_LIN_BUS_CONFLICT_CLOCK_DIV_1 | Bus conflict detection clock is base clock. |
| SCI_LIN_BUS_CONFLICT_CLOCK_DIV_2 | Bus conflict detection clock is base clock/2. |
| SCI_LIN_BUS_CONFLICT_CLOCK_DIV_4 | Bus conflict detection clock is base clock/4. |

### ◆ sci_lin_base_clock_t

| enum sci_lin_base_clock_t | |
|---|---|
| Asynchronous Mode Base Clock Select. The period of the SCI base clock is 1/16 of a single bit period when the SCIn.SEMR.ABCS is 0, and 1/8 of a single bit period when the SCIn.SEMR.ABCS is 1. | |
| Enumerator | |
| SCI_LIN_BASE_CLOCK_16 | Select 16 base clock cycles for 1-bit period. |
| SCI_LIN_BASE_CLOCK_8 | Select 8 base clock cycles for 1-bit period. |

## Function Documentation

### ◆ R_SCI_LIN_Open()

fsp_err_t R_SCI_LIN_Open ( lin_ctrl_t *const *p_api_ctrl*, lin_cfg_t const *const *p_cfg* )

Configures the LIN driver channel based on the input configuration.

Implements lin_api_t::open.

Example:

```
/* Open the LIN instance with initial configuration. */
fsp_err_t err = R_SCI_LIN_Open(&g_master_ctrl, &g_master_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or configuration structure is NULL. |
| FSP_ERR_INVALID_CHANNEL | The requested channel does not exist on this MCU or the channel does not support LIN mode. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_MODE | Setting not supported for selected mode |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_Write()

fsp_err_t R_SCI_LIN_Write ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking transmission of a LIN frame.

In master mode, the LIN header (break, sync and protected identifier) is sent before the LIN data.

- To transmit the LIN header only, set p_transfer_params->p_data to NULL.
- To transmit both header and data, set p_transfer_params->p_data to the TX buffer to transmit.

The LIN header is not transmitted in slave mode. Set p_transfer_params->p_data to the TX buffer to transmit.

On successful header-only transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_HEADER_COMPLETE. On successful data transmission or header+data transmission completion, the callback is called with event lin_event_t::LIN_EVENT_TX_DATA_COMPLETE.

Implements lin_api_t::write.

Example:

```
lin_transfer_params_t write_params =
   {
      .checksum_type = LIN_CHECKSUM_TYPE_ENHANCED,
      .id            = FRAME_ID,
      .num_bytes     = TRANSFER_LENGTH,
      .p_data        = g_tx_buf
   };
/* Send the LIN data. In master mode, the header (break, sync, and protected
identifier) will be sent before the data. */
   err = R_SCI_LIN_Write(&g_master_ctrl, &write_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission started successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, transfer parameters, or tx/rx buffer is NULL, or 0 bytes length provided |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SCI_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_Read()

fsp_err_t R_SCI_LIN_Read ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* )

Begins non-blocking data reception to receive user specified number of bytes into destination buffer pointer.

The checksum type specifies the checksum type used for validation. If a non-standard algorithm is used, or the application prefers to validate the checksum outside the driver, or the application prefers to skip checksum validation, specify lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE. If checksum validation is skipped, the lin_checksum_type_t::LIN_EVENT_ERR_INVALID_CHECKSUM event is not possible. When lin_checksum_type_t::LIN_CHECKSUM_TYPE_NONE is used, the number of bytes specified in the receive buffer length will be received (the driver will not expect to receive an additional 1 checksum byte), so if a non-standard checksum is used, sufficient space must be allocated in the write buffer and accounted for in the provided length.

On successful data reception completion, the callback is called with event lin_event_t::LIN_EVENT_RX_DATA_COMPLETE.

Implements lin_api_t::read.

Example:

```
/* Begin reception of the data */
   err = R_SCI_LIN_Read(&g_master_ctrl, &read_params);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception started successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block, transfer parameters, or tx/rx buffer is NULL, or 0 bytes length provided |
| FSP_ERR_IN_USE | A transmission or reception is currently in progress. Call R_SCI_LIN_CommunicationAbort to cancel it if desired, or wait for the current transfer operation to complete before starting a new one. |
| FSP_ERR_INVALID_CALL | Data reception is not possible because a header frame has not been received yet (slave mode only). |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_CommunicationAbort()

fsp_err_t R_SCI_LIN_CommunicationAbort ( lin_ctrl_t *const *p_api_ctrl*)

Cancels in progress frame data read or write, or header read or write. Break field reception cannot be cancelled. For slave nodes, reception of a new header is still enabled after a call to this function.

Implements lin_api_t::communicationAbort.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transfer aborted successfully or no transfer was in progress. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_CallbackSet()

fsp_err_t R_SCI_LIN_CallbackSet ( lin_ctrl_t *const *p_api_ctrl*, void(*)(lin_callback_args_t *) *p_callback*, void *const *p_context*, lin_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure.

Implements lin_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or callback is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_BaudCalculate()

fsp_err_t R_SCI_LIN_BaudCalculate ( sci_lin_baud_params_t const *const *p_baud_params*, sci_lin_baud_setting_t *const *p_baud_setting* )

Calculates baud rate and LIN timer (TCNT, TPRE and TCSS) register settings. This function evaluates and determines the most accurate settings for the baud rate and timer related registers.

The LIN timer setting is used for break field transmission/detection. Because the timer setting is specified in terms of bits, and the duration of a bit varies depending on baud rate, the baud rate register settings and timer register settings are related. The smallest possible LIN timer divider which can achieve the desired break field bits setting at the configured baud rate is selected to provide the highest measurement accuracy.

The baud rate cannot be updated at runtime with this function. This function is provided to ease configuration of the initial baud settings.

**Parameters**

| | | |
|---|---|---|
| [in] | p_baud_params | Parameters required to calculate the baud rate |
| [out] | p_baud_setting | If calculation succeeds, contains computed values to achieve requested baud rate. If calculation fails, the input structure is not modified. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Register settings updated in provided p_baud_setting |
| FSP_ERR_ASSERTION | p_baud_setting was NULL |
| FSP_ERR_INVALID_ARGUMENT | Cannot achieve combination of break field bits and baudrate with provided settings or p_baud_params->baudrate was 0 or p_baud_params->break_bits was 0 |

### ◆ R_SCI_LIN_IdFilterSet()

fsp_err_t R_SCI_LIN_IdFilterSet ( lin_ctrl_t *const *p_api_ctrl*, sci_lin_id_filter_setting_t const *const *p_config* )

Set the the ID filter settings for filtering control field 1 (PID byte).

NOTE: Setting the ID filter will abort any in-progress LIN header reception, as the ID filter settings cannot be changed during reception of the header. The next header will be received with the new settings.

```
/* Configure the ID filter */
   err = R_SCI_LIN_IdFilterSet(&g_slave_ctrl, &filter);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the LIN control block. |
| [in] | p_config | The ID filter settings to apply |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID filter updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or p_config is NULL. |
| FSP_ERR_INVALID_MODE | Function called by master node (not supported for master nodes) |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_IdFilterGet()

fsp_err_t R_SCI_LIN_IdFilterGet ( lin_ctrl_t *const *p_api_ctrl*, sci_lin_id_filter_setting_t *const *p_config* )

Returns the currently configured ID filter settings.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the LIN control block. |
| [out] | p_config | The current ID filter settings |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID filter updated successfully. |
| FSP_ERR_ASSERTION | Pointer to LIN control block or p_config is NULL. |
| FSP_ERR_INVALID_MODE | Function called by master node (not supported for master nodes) |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_Close()

fsp_err_t R_SCI_LIN_Close ( lin_ctrl_t *const *p_api_ctrl*)

Closes the LIN driver.

Implements lin_api_t::close.

Example:

```
/* Close the driver */
   err = R_SCI_LIN_Close(&g_master_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to LIN control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_WakeupSend()

fsp_err_t R_SCI_LIN_WakeupSend ( lin_ctrl_t *const *p_api_ctrl*)

Send wakeup is not supported on the SCI LIN driver.

Implements lin_api_t::wakeupSend.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_LIN_SleepEnter()

fsp_err_t R_SCI_LIN_SleepEnter ( lin_ctrl_t *const *p_api_ctrl*)

Sleep is not supported on the SCI LIN driver.

Implements lin_api_t::sleepEnter.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_LIN_SleepExit()

fsp_err_t R_SCI_LIN_SleepExit ( lin_ctrl_t *const *p_api_ctrl*)

Sleep is not supported on the SCI LIN driver.

Implements lin_api_t::sleepExit.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_SCI_LIN_InformationFrameRead()

fsp_err_t R_SCI_LIN_InformationFrameRead ( lin_ctrl_t *const *p_api_ctrl*, lin_transfer_params_t *const *p_transfer_params* )

[DEPRECATED] Use R_SCI_LIN_Read.

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_LIN_StartFrameWrite()

| fsp_err_t R_SCI_LIN_StartFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, uint8_t const *id* ) |
|---|
| [DEPRECATED] Use R_SCI_LIN_Write<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

### ◆ R_SCI_LIN_InformationFrameWrite()

| fsp_err_t R_SCI_LIN_InformationFrameWrite ( lin_ctrl_t *const *p_api_ctrl*, const lin_transfer_params_t *const *p_transfer_params* ) |
|---|
| [DEPRECATED] Use R_SCI_LIN_Write<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

## 5.2.6.20 SMBUS Communication Device (rm_comms_smbus)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | RM_COMMS_SMBUS_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg)<br><br>Opens and configures the SMBUS Comms module. Implements rm_comms_api_t::open. More... |
| fsp_err_t | RM_COMMS_SMBUS_Close (rm_comms_ctrl_t *const p_api_ctrl)<br><br>Disables specified SMBUS Comms module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_SMBUS_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)<br><br>Updates the SMBUS Comms callback. Implements rm_comms_api_t::callbackSet. More... |
| fsp_err_t | RM_COMMS_SMBUS_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |

| | |
|---|---|
| | Performs a read from the SMBUS device. Implements rm_comms_api_t::read. More... |
| fsp_err_t | RM_COMMS_SMBUS_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write from the SMBUS device. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_SMBUS_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs a write to, then a read from the SMBUS device. Implements rm_comms_api_t::writeRead. More... |
| void | rm_comms_smbus_transmission_callback (i2c_master_callback_args_t *p_args)<br>Common callback function called in the I2C driver callback function when SMBus is used. |
| void | rm_comms_smbus_timeout_callback (timer_callback_args_t *p_args)<br>Callback function called in the GPT driver callback function when SMBus is used. |

### Detailed Description

Middleware to implement the SMBUS communications interface. This module implements the Communications Middleware Interface.

# Overview

The RM_COMMS_SMBUS module implements COMMS API for SMBUS interface.

RM_COMMS_SMBUS supports SMBus protocol on following modules:

- R_IIC_MASTER
- R_IIC_B_MASTER

### Features

Supported SMBUS command:

- Send byte
- Receive byte
- Write byte
- Write word

- Read byte
- Read word
- Read block

Packet error check is supported by software CRC-8 (x^8 + x^2 + x + 1).

Supported RTOS (FreeRTOS and AzureOS).

# Configuration

### Build Time Configurations for rm_comms_smbus

The following build time configurations are defined in fsp_cfg/rm_comms_smbus_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Connectivity > SMBus Communication Device (rm_comms_smbus)

This module can be added to the Stacks tab via New Stack > Connectivity > SMBus Communication Device (rm_comms_smbus).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Manual Entry | g_comms_smbus0 | Module name. |
| Callback | Name must be a valid C symbol | comms_smbus_callback | A user callback function can be provided. |
| Semaphore Timeout (RTOS only) | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for semaphore operation in using RTOS. |
| Slave Address | Value must be non-negative | 0x00 | Specify the slave address. |
| CRC support | • Enable<br>• Disable | Enable | Use CRC-8 algorithm to generate PEC byte for SMBus communication. |

### Pin Configuration

This module uses SDA and SCL pins of I2C Master

# Usage Notes

If an RTOS is used, blocking and bus lock is available.

- If blocking of an SMBus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

If an RTOS is used and blocking and bus lock is enabled, RM_COMMS_SMBUS_Write(), RM_COMMS_SMBUS_Read() and RM_COMMS_SMBUS_WriteRead() cannot be called in callback.

Dependency module General PWM Timer (GPT) and Event Link Controller (ELC) need to be openned before opening SMBus comms device.

Protocol support by SMBUS API:

- RM_COMMS_SMBUS_Write(): Send byte, Write byte, Write word.
- RM_COMMS_SMBUS_Read(): Receive byte protocol.
- RM_COMMS_SMBUS_WriteRead(): Read byte, read word, block read.

Warning

The recommended frequency of PCLKB (when using r_iic_master) or I3CCLK (when using r_iic_b_master) is in range of 1 MHz - 86 MHz. If the frequency too high, the middleware will not able to meet the 300 ns of data hold time of SMBUS standard. On the contrary, the low frequency may cause the count of BRL smaller than SDA delay count which will lead to communication between devices might malfunction or falsely indicate a start or stop condition, depending on the bus state.

### Bus Initialization

The SMBUS communications interface expects a bus instance to be opened before opening any specific SMBUS comms device. The communications interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate SMBUS Communication Device (rm_comms_smbus) open call.

## Examples

### Basic Example

This is a basic example of minimal use of SMBus communications implementation in an application.

```
void rm_comms_smbus_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the relevant drivers if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_comms_i2c_extend =
       ((rm_comms_smbus_extended_cfg_t *)
(g_comms_smbus_cfg.p_extend))->p_comms_i2c_extend_cfg;
 i2c_master_instance_t * p_driver_iic   = (i2c_master_instance_t *)
p_comms_i2c_extend->p_driver_instance;
 elc_instance_t       * p_driver_elc   = (elc_instance_t *)
p_comms_i2c_extend->p_elc;
 timer_instance_t      * p_driver_timer = (timer_instance_t *)
p_comms_i2c_extend->p_timer;
```

```
    p_driver_iic->p_api->open(p_driver_iic->p_ctrl, p_driver_iic->p_cfg);
    p_driver_elc->p_api->open(p_driver_elc->p_ctrl, p_driver_elc->p_cfg);
    p_driver_timer->p_api->open(p_driver_timer->p_ctrl, p_driver_timer->p_cfg);
#if (BSP_CFG_RTOS != 0)
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_comms_i2c_extend->p_blocking_semaphore)
    {
 #if (BSP_CFG_RTOS == 1) // AzureOS

tx_semaphore_create(p_comms_i2c_extend->p_blocking_semaphore->p_semaphore_handle,

p_comms_i2c_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif (BSP_CFG_RTOS == 2) // FreeRTOS
       *(p_comms_i2c_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_comms_i2c_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_comms_i2c_extend->p_bus_recursive_mutex)
    {
 #if (BSP_CFG_RTOS == 1) // AzureOS
       tx_mutex_create(p_comms_i2c_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_comms_i2c_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif (BSP_CFG_RTOS == 2) // FreeRTOS
       *(p_comms_i2c_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_comms_i2c_extend->p_bus_recursive_
mutex->p_mutex_memory);
 #endif
    }
```

```
#endif
    err = RM_COMMS_SMBUS_Open(&g_comms_smbus_ctrl, &g_comms_smbus_cfg);
    assert(FSP_SUCCESS == err);
    err = RM_COMMS_SMBUS_CallbackSet(&g_comms_smbus_ctrl, comms_smbus_callback,
NULL);
    assert(FSP_SUCCESS == err);
    g_write_read_param.p_dest = g_read_buffer;
    g_write_read_param.p_src = g_write_buffer;
 /* SMBus send byte command */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    err = RM_COMMS_SMBUS_Write(&g_comms_smbus_ctrl, g_write_buffer, 1);
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
 /* SMBus receive byte command */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    err = RM_COMMS_SMBUS_Read(&g_comms_smbus_ctrl, g_read_buffer, 2); // 1 byte for
data, 1 byte for PEC.
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
 /* SMBus write byte command */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    g_write_buffer[0]          = (uint8_t) SMBUS_COMMAND_CODE;
    g_write_buffer[1]          = (uint8_t) SMBUS_DUMMY_WRITE_DATA;
    err = RM_COMMS_SMBUS_Write(&g_comms_smbus_ctrl, g_write_buffer, 2); // 1 byte for
command code, 1 byte for data. PEC byte will be padded automatically.
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
 /* SMBus read byte command */
```

```
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    g_write_buffer[0]          = (uint8_t) SMBUS_COMMAND_CODE;
    g_write_read_param.src_bytes = 1; // 1 byte of command code
    g_write_read_param.dest_bytes = 2; // 1 data byte + 1 PEC byte
    err = RM_COMMS_SMBUS_WriteRead(&g_comms_smbus_ctrl, g_write_read_param);
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
/* SMBus write word */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    g_write_buffer[0]          = (uint8_t) SMBUS_COMMAND_CODE;
    g_write_buffer[1]          = (uint8_t) SMBUS_DUMMY_WRITE_DATA; // Data byte low
    g_write_buffer[2]          = (uint8_t) SMBUS_DUMMY_WRITE_DATA; // Data byte high
    err = RM_COMMS_SMBUS_Write(&g_comms_smbus_ctrl, g_write_buffer, 3);
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
/* SMBus read word */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    g_write_buffer[0]          = (uint8_t) SMBUS_COMMAND_CODE;
    g_write_read_param.src_bytes = 1; // 1 byte of command code
    g_write_read_param.dest_bytes = 3; // 2 data byte + 1 PEC byte
    err = RM_COMMS_SMBUS_WriteRead(&g_comms_smbus_ctrl, g_write_read_param);
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
/* SMBus read block */
    g_smbus_callback_args.event = (rm_comms_event_t) INITIALIZE_EVENT;
    g_write_buffer[0]          = (uint8_t) SMBUS_COMMAND_CODE;
    g_write_read_param.src_bytes = 1; // 1 byte of command code
    g_write_read_param.dest_bytes = 10; // 1 block count byte + 8 data byte + 1 PEC
```

```
byte
    err = RM_COMMS_SMBUS_WriteRead(&g_comms_smbus_ctrl, g_write_read_param);
    assert(FSP_SUCCESS == err);
    wait_for_transmission();
    assert(RM_COMMS_EVENT_OPERATION_COMPLETE == g_smbus_callback_args.event);
    assert(RM_COMMS_SMBUS_NO_ERROR == g_smbus_error->smbus_event);
}
```

## Data Structures

| | |
|---|---|
| struct | rm_comms_smbus_error_t |
| struct | rm_comms_smbus_extended_cfg_t |
| struct | rm_comms_smbus_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rm_comms_smbus_event_t |

## Data Structure Documentation

### ◆ rm_comms_smbus_error_t

| struct rm_comms_smbus_error_t |
|---|
| SMBus error structure |

### ◆ rm_comms_smbus_extended_cfg_t

| struct rm_comms_smbus_extended_cfg_t | | |
|---|---|---|
| Extend configuration of SMBus | | |
| Data Fields | | |
| bool | pec_enable | Calculate PEC byte for SMBus transmission. |
| rm_comms_i2c_bus_extended_cfg_t * | p_comms_i2c_extend_cfg | Pointer to extend configuration block of rm_comms_i2c. |
| rm_comms_i2c_instance_ctrl_t * | p_comms_i2c_ctrl | Control block of rm_comms_i2c. |

### ◆ rm_comms_smbus_instance_ctrl_t

| struct rm_comms_smbus_instance_ctrl_t | | |
|---|---|---|
| SMBus middleware control block | | |
| Data Fields | | |
| bool | timer_is_enabled | Validate that external event |

| | | |
|---|---|---|
| | | triggers stop the timer is enabled. |
| uint8_t | write_buff[RM_COMMS_SMBUS_TRANSMISSION_MAX_BYTES] | Intermediate buffer. |
| uint8_t | receive_crc_seed | CRC seed value. |
| uint32_t | open | Open flag. |
| rm_comms_i2c_instance_ctrl_t * | p_comms_i2c_ctrl | Control block of rm_comms_i2c. |
| rm_comms_smbus_error_t * | p_smbus_error | SMBus specific error code. |
| void * | p_context | |

## Enumeration Type Documentation

### ◆ rm_comms_smbus_event_t

| enum rm_comms_smbus_event_t | |
|---|---|
| SMBus specific event | |
| Enumerator | |
| RM_COMMS_SMBUS_NO_ERROR | SMBus transmission complete without any error. |
| RM_COMMS_SMBUS_MISC_ERROR | Dependency modules failed. |
| RM_COMMS_SMBUS_DATA_CORRUPT | PEC byte is incorrect. |
| RM_COMMS_SMBUS_SEXT_TIMEOUT | Total transmission time exceeded 25 ms. |
| RM_COMMS_SMBUS_MEXT_TIMEOUT | Transmission time between each event exceeded 10 ms. |

## Function Documentation

### ◆ RM_COMMS_SMBUS_Open()

fsp_err_t RM_COMMS_SMBUS_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the SMBUS Comms module. Implements rm_comms_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Communications Middle module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SMBUS_Close()

fsp_err_t RM_COMMS_SMBUS_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified SMBUS Comms module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_SMBUS_CallbackSet()

fsp_err_t RM_COMMS_SMBUS_CallbackSet ( rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context* )

Updates the SMBUS Comms callback. Implements rm_comms_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_SMBUS_Read()

fsp_err_t RM_COMMS_SMBUS_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the SMBUS device. Implements rm_comms_api_t::read.

***Note***

*When Packet Error Check (PEC) is used, size of destination buffer and the number of reading bytes must have 1-byte in addition for PEC byte.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | Read data size is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SMBUS_Write()

fsp_err_t RM_COMMS_SMBUS_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write from the SMBUS device. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | Transfer data size is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SMBUS_WriteRead()

fsp_err_t RM_COMMS_SMBUS_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs a write to, then a read from the SMBUS device. Implements rm_comms_api_t::writeRead.

***Note***

*When Packet Error Check (PEC) is used, size of destination buffer and the number of reading bytes must have 1-byte in addition for PEC byte.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | Transfer data size is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.6.21 SMCI (r_sci_smci)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_SMCI_Open (smci_ctrl_t *const p_api_ctrl, smci_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_SMCI_Write (smci_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SCI_SMCI_Read (smci_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SCI_SMCI_TransferModeSet (smci_ctrl_t *const p_api_ctrl, smci_transfer_mode_t const *const p_transfer_mode_params) |
| fsp_err_t | R_SCI_SMCI_BaudSet (smci_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_SCI_SMCI_StatusGet (smci_ctrl_t *const p_api_ctrl, smci_status_t *const p_status) |
| fsp_err_t | R_SCI_SMCI_ClockControl (smci_ctrl_t *const p_api_ctrl, bool clock_enable) |

| | |
|---|---|
| fsp_err_t | R_SCI_SMCI_CallbackSet (smci_ctrl_t *const p_api_ctrl, void(*p_callback)(smci_callback_args_t *), void *const p_context, smci_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_SMCI_Close (smci_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_SMCI_BaudCalculate (smci_speed_params_t const *const p_speed_params, uint32_t baud_rate_error_x_1000, void *const p_baud_setting) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SMCI Interface.

# Overview

### Features

The SCI SMCI module supports the following features:

- Abort in-progress read/write operations
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, RX char, error, etc)
- Transfer mode (normal and block) and data convention type change at run time.
- All channels with SMCI support shall be configurable by the driver
- Baud-rate (and ETU) calculation and change based on ISO7816 asynchronous parameters at run-time. ETU is 1/baud.
- Error notifications of parity error, error signal reception (guard time) and overrun.
- Clock output control

# Configuration

### Build Time Configurations for r_sci_smci

The following build time configurations are defined in fsp_cfg/r_sci_smci_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Connectivity > SMCI (r_sci_smci)

This module can be added to the Stacks tab via New Stack > Connectivity > SMCI (r_sci_smci). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_smci0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SCI channel. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 9600 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated baud_setting_t structure. |
| Clock conversion integer | Refer to the RA Configuration tool for available options. | F=372 fmax=5MHz D=1 | Clock rate conversion integer is one of transmission parameters defined by ISO7816-3 as F and also decodes to f-max of the output clock. |
| Max Error (%) | Must be a valid non negative integer with a maximum configurable value of 100 | 5 | Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings.<br><br>NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated baud_setting_t structure. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | user_smci_callback | A user callback function can be |

| | | |
|---|---|---|
| | | provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | Select the receive interrupt priority. |
| Transmit Data Empty Interrupt Priority | MCU Specific Options | Select the transmit interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | Select the error interrupt priority. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA2A2 | PCLKB |
| RA2E1 | PCLKB |
| RA2E2 | PCLKB |
| RA2E3 | PCLKB |
| RA2L1 | PCLKB |
| RA2L2 | PCLKB |
| RA2T1 | PCLKB |
| RA4C1 | PCLKA |
| RA4E1 | PCLKA |
| RA4E2 | PCLKA |
| RA4L1 | PCLKA |
| RA4M1 | PCLKA |
| RA4M2 | PCLKA |
| RA4M3 | PCLKA |
| RA4T1 | PCLKA |
| RA4W1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M1 | PCLKA |

| | |
|---|---|
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T1 | PCLKA |
| RA6T3 | PCLKA |

The clock source for the baud-rate clock generator is the PCLK. It is scaled by the SMR_SMCI.CKS bits to achieve the requested baud rate. This is done in R_SCI_SMCI_BaudSet routine.

**Pin Configuration**

This module uses TXDx and RXDx to communicate to external devices. TXDx and RXDx need to be tied together and pulled up to VCC externally via a pullup resistor and connected to the DATA line of a connected card.

The SmartCard clock signal is generated by the SMCI module on the SCKx pin. The clock frequency produced by the module is (baudrate * F) / D. The ISO specification defines the valid maximum frequency range as 4Mhz to 20MHz; only certain combinations of Fi, Di and f(max) are allowed.

When writing the application for the driver, the application developer must also allocate a software controlled Reset line via a GPIO. This will allow the reliable receipt of the ATR message. Optionally, VCC and VPP can also be controlled by a GPIO output, so that cold starts can be forced.

# Usage Notes

The SMCI module is compliant to ISO7816-3. SMCI is a half duplex interface. Direct convention in T0 mode is the default. The driver supports both direct and indirect (inverted) modes of transmission. It also supports GSM mode in which the output clock can be enabled and disabled while the interface is still active. If it is known that the device connected is a device in inverse convention, the convention can be changed with smci_api_t::transferModeSet() after calling open.

The MCU creates the clock for the attached SMCI device based on the initial baudrate setting. Upon reset the SmartCard device advertises the Di and Fi parameters as well as the max clock speed it can handle. The Di and Fi parameters dictate at what rate the SmartCard device can sample the data line as a function of the supplied clock generated by the RA MCU. Only certain combinations of D and F are supported by the SMCI on SCI module. Only combinations where the the ratio of S=F/D corresponds to a value of S of 32, 64, 93, 128, 186, 256, 372, or 512 (for example see Table "Base clock settings in smart card interface mode" in the SCI section of the relevant hardware manual).

The baud rate (1/ETU) can be changed while the device is open to allow for speed negotiation based on the attached device's capabilities.

The SMCI module does not contain a FIFO, and as such the receipt of multi-byte data has to be handled by interrupt initiated callback. The application developer must develop their callback so that the receipt of data is handled sufficiently without receiver overrun. If the read routine is called with a length=0, every receive interrupt will initiate a call to the user's callback. If the read is called with a non-zero length... the interrupt will fill the user's read buffer and initiate the callback after the last byte is complete. In many cases, the user can send an event from their callback so that the reading routine can wait for the event with a timeout. If a timeout occurs, the user can return the read state machine to an idle state by calling the read routine with a length of 0.

### Limitations

# Examples

### SCI SMCI Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_smci_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the smci instance with initial configuration. */
 fsp_err_t err = R_SCI_SMCI_Open(&g_smci0_ctrl, &g_smci0_cfg);
    assert(FSP_SUCCESS == err);
 /* Need to have clock on inorder to receive or transmit*/
 R_SCI_SMCI_ClockControl(&g_smci0_ctrl, true);
    err = R_SCI_SMCI_Read(&g_smci0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_SMCI_Write(&g_smci0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
}
void user_smci_callback (smci_callback_args_t * p_args)
```

```
{
 /* Handle the SMCI event */
 switch (p_args->event)
    {
 /* Received a character */
 case SMCI_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
                g_out_of_band_received[g_out_of_band_index++] = p_args->data;
      }
 break;
      }
 /* Receive complete */
 case SMCI_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
 /* Transmit complete */
 case SMCI_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
    }
}
```

### SCI SMCI Baud Set Example

```
#define SCI_SMCI_BAUDRATE_28800 (28800)
#define SCI_SMCI_BAUDRATE_ERROR_PERCENT_5 (5000)
void r_sci_smci_baud_example (void)
{
 smci_speed_params_t speed_settings;
 smci_baud_setting_t baud_setting;
    speed_settings.baudrate = SCI_SMCI_BAUDRATE_28800;
    speed_settings.fi       = SMCI_CLOCK_CONVERSION_INTEGER_512_5;
    speed_settings.di       = SMCI_BAUDRATE_ADJUSTMENT_INTEGER_4;
 fsp_err_t err = R_SCI_SMCI_BaudCalculate(&speed_settings,
SCI_SMCI_BAUDRATE_ERROR_PERCENT_5, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SCI_SMCI_BaudSet(&g_smci0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | sci_smci_instance_ctrl_t |
| struct | smci_baud_setting_t |
| struct | sci_smci_extended_cfg_t |

## Data Structure Documentation

### ◆ sci_smci_instance_ctrl_t

| struct sci_smci_instance_ctrl_t |
|---|
| SMCI instance control block. |

### ◆ smci_baud_setting_t

| struct smci_baud_setting_t | | |
|---|---|---|
| Register settings to achieve a desired baud rate in Smart Card mode | | |
| Data Fields | | |
| uint32_t | computed_baud_rate | |
| union smci_baud_setting_t | __unnamed__ | |
| uint8_t | scmr_bcp2: 1 | BCP2 setting in Smart Card Mode Register. |

| | | |
|---|---|---|
| uint8_t | brr | Bit Rate Register setting. |

### ◆ sci_smci_extended_cfg_t

| struct sci_smci_extended_cfg_t | | |
|---|---|---|
| SMCI on SCI device Configuration | | |
| Data Fields | | |
| smci_baud_setting_t * | p_smci_baud_setting | Register settings for a desired baud rate. |

## Function Documentation

### ◆ R_SCI_SMCI_Open()

fsp_err_t R_SCI_SMCI_Open ( smci_ctrl_t *const *p_api_ctrl*, smci_cfg_t const *const *p_cfg* )

Configures the Smart Card Interface driver based on the input configurations. The interface stays in the clock-off state without enabling reception at the end of this function. ISO7816-3 default communication parameters are used to initial ize SMCI port speed and parameters, as the ATR message is always sent in that format. Only if Inverse convention is expected should the transfer mode be changed after reset. Implements smci_api_t::open

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block that is to be opened |
| [in] | p_cfg | Pointer to the config structure that shall be used to set parameters of the SMCI baud calculations needed to be have done and set into p_cfg->p_extend->p_smci_baud_setting |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_SMCI_Write()

fsp_err_t R_SCI_SMCI_Write ( smci_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements smci_api_t::write

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block that is to be opened |
| [in] | p_src | Pointer to buffer that will be written out |
| [in] | bytes | Number of bytes to be transferred |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission started successfully. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A SMCI transmission is in progress |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_SMCI_Read()

fsp_err_t R_SCI_SMCI_Read ( smci_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Receiving is done at the isr level as there is no FIFO. If 0 is passed in as the length, reception will always invoke the user callback. Implements smci_api_t::read

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block that is to be opened |
| [in,out] | p_dest | Pointer to the buffer top be read into |
| [in] | bytes | Number of bytes to copy from the SMCI receive register |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block or read buffer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_SMCI_TransferModeSet()

fsp_err_t R_SCI_SMCI_TransferModeSet ( smci_ctrl_t *const *p_api_ctrl*, smci_transfer_mode_t const *const *p_transfer_mode_params* )

Updates the settings of block transfer mode and data transfer convention. The SCMR and SMR_SMCI registers will be set according to the input arguments of protocol type, data convention type, and mode. Implements smci_api_t::transferModeSet

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block that is to be modified |
| [in] | p_transfer_mode_params | Pointer to SMCI settings like protocol, convention, and gsm_mode |

**Warning**

This terminates any in-progress transmission and reception.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer mode and data transfer direction was successfully changed. |
| FSP_ERR_IN_USE | Unable to change transfer mode as device has clock off or is actively RX or TX |
| FSP_ERR_ASSERTION | Null pointer was passed as a parameter |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_SCI_SMCI_BaudSet()

fsp_err_t R_SCI_SMCI_BaudSet ( smci_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate and clock output. p_baud_setting is a pointer to a smci_baud_setting_t structure that needs to have already been filled by R_SCI_SMCI_BaudCalculate Implements smci_api_t::baudSet

**Warning**

This terminates any in-progress transmission.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block that is to be modified |
| [in] | p_baud_setting | Pointer to baud setting information to be written to the SMCI hardware registers |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block or p_baud_setting is NUL |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_INVALID_ARGUMENT | The p_baud_setting does not seem to be set correctly |

### ◆ R_SCI_SMCI_StatusGet()

fsp_err_t R_SCI_SMCI_StatusGet ( smci_ctrl_t *const *p_api_ctrl*, smci_status_t *const *p_status* )

Provides the state of the driver and the # of bytes received since read was called Implements smci_api_t::statusGet

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to SMCI control block of this SMCI instance |
| [out] | p_status | Pointer structure that will be filled in with status info |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block, or info structure is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_SMCI_ClockControl()

fsp_err_t R_SCI_SMCI_ClockControl ( smci_ctrl_t *const *p_api_ctrl*, bool *clock_enable* )

Enable or disable the clock signal that is provided by interface the baud rate. When the clock is enabled, reception is enabled at the end of this function. Clock output control as defined in "Clock Output Control" of "Operation in Smart Card Interface Mode" in the SCI section of the relevant hardware manual. Implements smci_api_t::clockControl

**Warning**

This terminates any in-progress transmission and reception.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_api_ctrl | Pointer to SMCI control block |
| [in] | clock_enable | true=Enable or false=disable the Smart Card Interface clock |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock output setting was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block is NULL |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_INVALID_MODE | Clock cannot be disabled if GSM mode isnt active |

### ◆ R_SCI_SMCI_CallbackSet()

fsp_err_t R_SCI_SMCI_CallbackSet ( smci_ctrl_t *const *p_api_ctrl*, void(*)(smci_callback_args_t *) *p_callback*, void *const *p_context*, smci_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements smci_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_SMCI_Close()

fsp_err_t R_SCI_SMCI_Close ( smci_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Implements smci_api_t::close

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to SMCI control block that is reqested to close |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to SMCI control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_SMCI_BaudCalculate()

fsp_err_t R_SCI_SMCI_BaudCalculate ( smci_speed_params_t const *const *p_speed_params*, uint32_t *baud_rate_error_x_1000*, void *const *p_baud_setting* )

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers. And then updates the SCI registers.

**Parameters**

| | | |
|---|---|---|
| [in] | p_speed_params | structure including speed defining paramets, baud, F, D, and max frequency |
| [in] | baud_rate_error_x_1000 | <baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 20000 (20%) according to the ISO spec. |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate setting calculation successful |
| FSP_ERR_ASSERTION | p_speed params or p_baud is a null pointer |
| FSP_ERR_INVALID_ARGUMENT | Baud rate is '0', freq is '0', or error in calculated baud rate is larger than 20%. |

### 5.2.6.22 SPI (r_sau_spi)

Modules » Connectivity

#### Functions

| | |
|---|---|
| fsp_err_t | R_SAU_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SAU_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SAU_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SAU_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SAU_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SAU_SPI_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_SPI_CalculateBitrate (uint32_t bitrate, sau_spi_div_setting_t *sclk_div, uint8_t sau_unit, uint8_t channel) |

#### Detailed Description

Driver for the SAU peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - Single Transfer Mode or Continuous Transfer Mode
  - Data Phase (DAPmn)
    - DAPmn=0 Data output starts from the start of the operation of the serial clock
    - DAPmn=1 Data output starts half a clock cycle before the start of the serial clock operation
  - Clock Phase (CKPmn)
    - CKPmn=0 Non-reverse
    - CKPmn=1 Reverse
  - MSB/LSB first
- DTC Support
- Callback Events
  - Transfer Complete

# Configuration

- RX Overflow Error (The SAU shift register is copied to the data register before previous data was read)

**Build Time Configurations for r_sau_spi**

The following build time configurations are defined in fsp_cfg/r_sau_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Critical Section Guarding | • Enabled<br>• Disabled | Disabled | Enable critical section guarding around peripheral configuration updates. This should be enabled if the R_SAU_I2C or R_SAU_UART module is being used simultaneously with this module. |
| Enable Single Channel | MCU Specific Options | | Enable single channel to reduce code size if only one channel is to be configured for SAU SPI. |
| Transfer Operating Mode | • Reception<br>• Transmission<br>• Transmission/ Reception | Transmission/Reception | Select transfer operation mode. |
| DTC Support | • Enabled<br>• Disabled | Disabled | Enable DTC support for the SAU SPI module. |
| Multiple Slave Compatibility Mode | • Enabled<br>• Disabled | Disabled | Enable multiple slave compatibility mode when SAU SPI is configured to slave mode for multiple-slave applications. |
| Continuous Transfer Mode | • Enabled<br>• Disabled | Enabled | Set to 'Enabled' if transfer in continuous mode is needed. |

### Configurations for Connectivity > SPI (r_sau_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sau_spi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | MCU Specific Options | | Select the SAU channel. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Operation Clock | • CKm0<br>• CKm1 | CKm0 | Select the operation clock. Use the Clocks tab to set the operation clock divider. |
| Transfer Mode | • Single transfer mode<br>• Continuous transfer mode | Single transfer mode | Select transfer mode in transfer end interrupt. But buffer empty interrupt (in continuous transfer mode) cannot be selected in Slave Reception. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select of data transfer sequence. |
| Data Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select when data output shall start compared with the serial clock operation. |
| Clock Phase | • High when idle<br>• Low when idle | High when idle | Select clock phase. |
| Bitrate | Value must be an integer greater than 0 | 500000 | Enter the desired bitrate.<br><br>If the requested bitrate cannot be achieved, adjust the operation clock frequency until the |

| | | | |
|---|---|---|---|
| | | | bitrate is achievable. The calculated bitrate is printed in a comment in the generated sau_spi_extended_cfg_t structure. |
| Callback | Name must be a valid C symbol | sau_spi_callback | A user callback function that is called from the sau spi interrupts when a transfer is completed or an error has occurred. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |

**Clock Configuration**

The SAU clock uses the system clock (ICLK) as its clock source.

A prescaler is applied to the ICLK in order to produce the operation clock frequency. The operation clock is used to generate the desired transfer period of the SAU module.

SAU operation clocks are shared among all channels within a SAU unit. Check the Hardware User's Manual for your MCU for available units and channels. SAU operation clock dividers are configurable in the Clocks tab.

The operation clock dividers are named SAU CKmn where m is the SAU unit, and n is the operation clock. For example, SAU CK01 applies to all SAU0 instances using CK1 as the operation clock (m=0, n=1).

**Clock Phase/Polarity Configuration**

The following table illustrates the settings of the Clock Phase/Polarity corresponding SCRmn register DCP[1:0] bits in the SAU SPI.

| Clock Phase | Clock Polarity | DCP[1:0] Value |
|---|---|---|
| Data sampling on odd edge, data variation on even edge | High when idle | 0b00 |
| Data sampling on odd edge, data variation on even edge | Low when idle | 0b01 |
| Data sampling on even edge, data variation on odd edge | High when idle | 0b10 |
| Data sampling on even edge, data variation on odd edge | Low when idle | 0b11 |

**Pin Configuration**

This module uses SCKmn, SOmn, and SImn pins to communicate with on board devices.

*Note*

*It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.*

### Enabling DTC with the SAU SPI

- DTC transfer is disabled by default. When DTC is enabled, error event will not be handled until transfer is completed.
- For further details on DTC please refer Transfer (r_dtc)

# Usage Notes

### Enabling Single Channel By User With The SAU SPI

- The Common > Single Channel property is used for reducing code size when only 1 SAU channel is to be configured for SAU SPI.
- Single Channel is configurable and disabled by default in the driver code.
- Note: Not all SAU channels are available on all pin layouts. Check the Hardware User Manual for your device to confirm available function assignment for each SAU channel.

### Transfer Complete Event

The transfer complete event is triggered when all of the data has been transferred. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

### Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in RX Overflow errors.

To improve performance at high bit rates, it is recommended that the instance be configured to service transfers using the DTC.

### Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

### Single Channel Use Case

If only SAU channel 00 is to be used for I2C or SPI or UART, enable single channel can reduce the code size.

### Continuous Transfer Mode

- In SAU SPI slave mode, the maximum bit rate of continuous transmission is up to 2.5 MHz.
- If only single transfer mode is to be used for SAU SPI, disabling 'Common > Continuous Transfer Mode' can reduce the code size.

### Selecting Operation Clock Frequency

The relationship between operation clock frequency and bitrate is: bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ] where:

- SDRmn.STCLK is an integer in the range [0, 127] for SAU SPI
- f_mck is the operation clock (SAU CKmn) frequency

By plugging in the minimum and maximum SDRmn.STCLK values, the range of bitrates for a given operation clock frequency can be obtained.

Note that due to STCLK being set as discrete integers, the actual bitrate may not be exact. The actual bitrate and percent errors can be calculated by the formulas:

```
actual_bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ]

percent_error = 100 * abs [(actual_bitrate - expected_bitrate) / expected_bitrate]
```

Using the fastest possible operation clock for the desired bitrate will result in the lowest deviation from the requested bitrate. Set the CKmn operation clock divider in the Clocks tab to select the desired operation clock frequency.

### Runtime bitrate calculation

The function R_SAU_SPI_CalculateBitrate can be used at runtime to calculate alternate bitrate settings.

This function computes settings with both operation clocks CK0 and CK1. If valid settings are possible with both clocks, it selects the settings and clock that would produce the lowest error.

Set the divisors for CK0 and CK01 in the clocks tab such that all required bitrate settings for the application are possible. A large range of bitrates can be achieved by having one "slow" operation clock for low speed modes and one "fast" operation clock for high speed modes.

### Limitations

- When multiple channels on the same SAU unit need to be used, and one is configured as I2C, then critical section property needs to be enabled.

## Examples

### Basic Example

This is a basic example of minimal use of the SAU_SPI in an application.

```
static volatile bool g_transfer_complete = false;

static void r_sau_spi_callback (spi_callback_args_t * p_args)

{

 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
```

```
    {
       g_transfer_complete = true;
    }
}
void sau_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SAU_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SAU_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
```

```
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SAU_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

| | |
|---|---|
| #define | SAU_SPI_TRANSFER_MODE_RECEPTION<br>Reception only. |
| #define | SAU_SPI_TRANSFER_MODE_TRANSMISSION<br>Transmission only. |
| #define | SAU_SPI_TRANSFER_MODE_TRANSMISSION_RECEPTION<br>Transmission/reception. |
| enum | sau_spi_operation_clock_t |
| enum | sau_spi_transfer_mode_t |
| enum | sau_spi_data_phase_t |
| enum | sau_spi_clock_phase_t |

## Enumeration Type Documentation

### ◆ sau_spi_operation_clock_t

| enum sau_spi_operation_clock_t |
|---|
| Selection of operating clock (fMCK) of channel |

### ◆ sau_spi_transfer_mode_t

| enum sau_spi_transfer_mode_t |
|---|
| Selection of transfer mode of channel |

### ◆ sau_spi_data_phase_t

| enum sau_spi_data_phase_t |
|---|
| Data phase |

### ◆ sau_spi_clock_phase_t

| enum sau_spi_clock_phase_t |
|---|
| Clock phase |

## Function Documentation

### ◆ R_SAU_SPI_Open()

fsp_err_t R_SAU_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SAU channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to the control structure. |
| p_cfg | Pointer to a configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

### ◆ R_SAU_SPI_Read()

fsp_err_t R_SAU_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Data frame length (Set to SPI_BIT_WIDTH_7_BITS or SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Bit width is not 8 bits<br>• Length is equal to 0<br>• Pointer to destination is NULL |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SAU_SPI_Write()

fsp_err_t R_SAU_SPI_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Data frame length (Set to SPI_BIT_WIDTH_7_BITS or SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SAU_SPI_WriteRead()

fsp_err_t R_SAU_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Data frame length (Set to SPI_BIT_WIDTH_7_BITS or SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Pointer to destination is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SAU_SPI_CallbackSet()

fsp_err_t R_SAU_SPI_CallbackSet ( spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_SAU_SPI_Close()

fsp_err_t R_SAU_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

Disable the SAU channel and set the instance as not open. Implements spi_api_t::close.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to an opened instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SAU_SPI_CalculateBitrate()

fsp_err_t R_SAU_SPI_CalculateBitrate ( uint32_t *bitrate*, sau_spi_div_setting_t * *sclk_div*, uint8_t *sau_unit*, uint8_t *channel* )

Calculate the register settings required to achieve the desired bitrate.

***Note***

*This function calculates the bitrate settings with both operation clocks CK0 and CK1, then selects the operation clock and register setting combination that would produce the lowest error.*

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | bitrate [bps]. For example, 250,000; 500,00; 16,000,000 (max), etc. |
| [out] | sclk_div | Pointer to sau_spi_div_setting_t used to configure baudrate settings. |
| | sau_unit | SAU unit. |
| | channel | SAU channel. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Bitrate is calculated successfully. |
| FSP_ERR_ASSERTION | Bitrate is not achievable or not valid for the selected unit/channel. |

## 5.2.6.23 SPI (r_sci_b_spi)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_B_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_B_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_B_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_B_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_B_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, |

| | |
|---|---|
| | void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_B_SPI_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_SPI_CalculateBitrate (uint32_t bitrate, sci_b_spi_clock_source_t clock_source, sci_b_spi_div_setting_t *sclk_div) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Select data sampling on leading edge, data change on trailing edge
    - CPHA=1 Select data change on leading edge, data sampling on trailing edge
  - MSB/LSB first
- Configurable bit rate
- DTC Support (with or without FIFO mode)
- DMAC Support (with or without FIFO mode)
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

# Configuration

### Build Time Configurations for r_sci_b_spi

The following build time configurations are defined in fsp_cfg/r_sci_b_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable Support for using a Transfer API | • Enabled<br>• Disabled | Enabled | Select whether to include support for DMAC or DTC transfers |

| | | | |
|---|---|---|---|
| | | | in the build. |
| FIFO Support | • Enabled<br>• Disabled | Disabled | Enable FIFO support for the SCI_B_SPI module. |

**Configurations for Connectivity > SPI (r_sci_b_spi)**

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sci_b_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_sci_spi0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SCI channel. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |
| Clock Source | • PCLK<br>• SCISPICLK | PCLK | Select whether the peripheral clock (PCLK) or SCISPICLK is used for generating the SCK frequency. |
| Callback | Name must be a valid C symbol | sci_b_spi_callback | A user callback function that is called from the sci_b_spi interrupts when a transfer is completed or an error has occurred. |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |

| | | | |
|---|---|---|---|
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |
| Bitrate | Value must be an integer greater than 0 | 8000000 | Enter the desired bitrate.<br><br>If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The actual bitrate is printed in a comment in the generated sci_spi_extended_cfg_t structure. |
| Transmit FIFO Trigger Level | Refer to the RA Configuration tool for available options. | Disabled | Set to 1 to trigger DMAC/DTC TX FIFO reload immediately when each byte is transmitted. Set to 15 to trigger the reload when the FIFO is full or after 15 bit times with no data. |
| Receive Sampling Timing Adjustment (Master Only) | • Default (No Delay)<br>• 1 TCLK Cycle<br>• 2 TCLK Cycles<br>• 3 TCLK Cycles<br>• 4 TCLK Cycles | Default (No Delay) | Select the sampling timing delay of the RXDn pin (master mode only). Configuring this property may be required at high bitrates for the master to sample the received data accurately. This setting has no effect in slave mode. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA6T2 | PCLKA |
| RA8D1 | PCLKA |
| RA8D2 | PCLKA |

| | |
|---|---|
| RA8E1 | PCLKA |
| RA8E2 | PCLKA |
| RA8M1 | PCLKA |
| RA8M2 | PCLKA |
| RA8P1 | PCLKA |
| RA8T1 | PCLKA |
| RA8T2 | PCLKA |

The SCI peripheral uses the SCISPICLK/SCICLK or PCLKA for communication and PCLKA for internal operations. Both can be configured via the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

*Note*

*It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.*

# Usage Notes

### Transfer Complete Event

The transfer complete event is triggered when all of the data has been transferred. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

### Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. This may also result in SPI_EVENT_ERR_READ_OVERFLOW errors.

In order to improve performance at high bit rates, consider the following recommendations:

- Enable DTC or DMAC. Note that DMAC will provide the best performance.
- Enable the Transmit Fifo Trigger in the module properties.
- Configure the Receive Sampling Timing Adjustment in the module properties (master mode only). For more information about Receive Sampling Timing Adjustment, refer to the diagram "Reception Sampling Timing Adjustment Operation in Clock Synchronous Mode (Master)" in the HW User's Manual for the selected MCU.
- When using DMAC, set the RX DMAC channel to a higher priority (lower channel number) than the TX channel.
- When using DMAC, consider pausing operation of higher-priority DMAC instances in the project during SPI transfers or reassigning priorities to ensure high-speed SPI channels have top priority to avoid bus contention.

### Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC or DMAC is not being used.

### Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

## Examples

### Basic Example

This is a basic example of minimal use of the SCI_B_SPI module in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_b_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_b_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SCI_B_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
```

```
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

## Data Structures

| | |
|---|---|
| struct | sci_b_spi_div_setting_t |

## Data Structure Documentation

### ◆ sci_b_spi_div_setting_t

| struct sci_b_spi_div_setting_t |
| --- |
| Settings for adjusting the SPI CLK. |

### Function Documentation

#### ◆ R_SCI_B_SPI_Open()

fsp_err_t R_SCI_B_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

**Parameters**

| | |
| --- | --- |
| p_api_ctrl | Pointer to the control structure. |
| p_cfg | Pointer to a configuration structure. |

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |
| FSP_ERR_UNSUPPORTED | Fifo mode is requested but a transfer interface is not used |

#### ◆ R_SCI_B_SPI_Read()

fsp_err_t R_SCI_B_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.

- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Bit width is not 8 bits<br>• Length is equal to 0<br>• Pointer to destination is NULL<br>• Transfer length is larger than supported by the transfer instances. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_B_SPI_Write()

fsp_err_t R_SCI_B_SPI_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits<br>• Transfer length is larger than supported by the transfer instances. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_B_SPI_WriteRead()

fsp_err_t R_SCI_B_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Pointer to destination is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits<br>• Transfer length is larger than supported by the transfer instances. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |

| | |
|---|---|
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_B_SPI_CallbackSet()

fsp_err_t R_SCI_B_SPI_CallbackSet ( spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_B_SPI_Close()

fsp_err_t R_SCI_B_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to an opened instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SCI_B_SPI_CalculateBitrate()

fsp_err_t R_SCI_B_SPI_CalculateBitrate ( uint32_t *bitrate*, sci_b_spi_clock_source_t *clock_source*, sci_b_spi_div_setting_t * *sclk_div* )

Calculate the register settings required to achieve the desired bitrate.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc. |
| | clock_source | clock source (PCLKA or SCISPICLK) used for bit rate calculation. |
| | sclk_div | Pointer to sci_b_spi_div_setting_t used to configure baudrate settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully. |
| FSP_ERR_ASSERTION | Baud rate is not achievable. |

***Note***

*The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.*

## 5.2.6.24 SPI (r_sci_spi)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SCI_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |

| | |
|---|---|
| fsp_err_t | R_SCI_SPI_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_SPI_CalculateBitrate (uint32_t bitrate, sci_spi_div_setting_t *sclk_div, bool use_mddr) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Select data sampling on leading edge, data change on trailing edge
    - CPHA=1 Select data change on leading edge, data sampling on trailing edge
  - MSB/LSB first
- Configurable bit rate
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

# Configuration

### Build Time Configurations for r_sci_spi

The following build time configurations are defined in fsp_cfg/r_sci_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC Support | • Enabled<br>• Disabled | Enabled | If support for transferring data using the DTC will be compiled in. |

### Configurations for Connectivity > SPI (r_sci_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sci_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SCI channel. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |
| Callback | Name must be a valid C symbol | sci_spi_callback | A user callback function that is called from the sci spi interrupts when a transfer is completed or an error has occurred. |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |
| Bitrate | Value must be an integer greater than 0 | 8000000 | Enter the desired bitrate.<br><br>If the requested bitrate cannot be achieved, the settings with the |

| | | | |
|---|---|---|---|
| | | | largest possible value that is less than or equal to the requested bitrate is used. The actual bitrate is printed in a comment in the generated sci_spi_extended_cfg_t structure. |
| Bitrate Modulation | • Disabled<br>• Enabled | Disabled | Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA2A2 | PCLKB |
| RA2E1 | PCLKB |
| RA2E2 | PCLKB |
| RA2E3 | PCLKB |
| RA2L1 | PCLKB |
| RA2L2 | PCLKB |
| RA2T1 | PCLKB |
| RA4C1 | PCLKA |
| RA4E1 | PCLKA |
| RA4E2 | PCLKA |
| RA4L1 | PCLKA |
| RA4M1 | PCLKA |
| RA4M2 | PCLKA |
| RA4M3 | PCLKA |
| RA4T1 | PCLKA |

| | |
|---|---|
| RA4W1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T1 | PCLKA |
| RA6T3 | PCLKA |

### Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

*Note*

*It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.*

# Usage Notes

### Transfer Complete Event

The transfer complete event is triggered when all of the data has been transferred. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

### Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in RX Overflow errors. In order to improve performance at high bit rates, it is recommended that the instance be configured to service transfers using the DTC.

Also note when using DTC for data transfer, SCI SPI can experience overrun error if the application makes heavy use of DMAC and/or another DTC instance. This is because DMAC has a higher priority over DTC in arbitration for the mastership of the DMA bus, and the arbitration between different triggers/transfers in DTC is done based on interrupt/trigger priority. Overrun error can be averted with one of the following options:

1. If SCI SPI uses DTC for data transfer, avoid the use of DMAC or another DTC instance in an application.
2. If DMAC/DTC is a must for other data transfers, avoid the use of DTC with SCI SPI.
3. Use the RSPI instead of SCI SPI. The RSPI hardware can handle overrun conditions.

### Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI_SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC is not being used.

### Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

### Bit Rate Modulation

Depending on the peripheral clock frequency, the desired bit rate may not be achievable. With bit rate modulation, the device can remove a configurable number of input clock pulses to the internal bit rate counter in order to create the desired bit rate. This has the effect of changing the period of individual bits in order to achieve the desired average bit rate. For more information see section 34.9 Bit Rate Modulation Function in the RA6M3 manual.

## Examples

### Basic Example

This is a basic example of minimal use of the SCI_SPI in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SCI_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

## Data Structures

| | |
|---|---|
| struct | sci_spi_div_setting_t |

## Data Structure Documentation

### ◆ sci_spi_div_setting_t

| struct sci_spi_div_setting_t | | |
|---|---|---|
| Settings for adjusting the SPI CLK. | | |
| Data Fields | | |
| uint8_t | brr | |
| uint8_t | cks: 2 | |
| uint8_t | mddr | Set to 0 to disable MDDR. |

## Function Documentation

### ◆ R_SCI_SPI_Open()

fsp_err_t R_SCI_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to the control structure. |
| p_cfg | Pointer to a configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

### ◆ R_SCI_SPI_Read()

fsp_err_t R_SCI_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Bit width is not 8 bits<br>• Length is equal to 0<br>• Pointer to destination is NULL |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_Write()

fsp_err_t R_SCI_SPI_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_WriteRead()

fsp_err_t R_SCI_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | Pointer to the control structure. |
| | p_src | Pointer to the source buffer. |
| | p_dest | Pointer to the destination buffer. |
| [in] | length | The number of bytes to transfer. |
| [in] | bit_width | Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | One of the following invalid parameters passed:<br>• Pointer p_api_ctrl is NULL<br>• Pointer to source is NULL<br>• Pointer to destination is NULL<br>• Length is equal to 0<br>• Bit width is not equal to 8 bits |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_UNSUPPORTED | The given bit_width is not supported. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

### ◆ R_SCI_SPI_CallbackSet()

fsp_err_t R_SCI_SPI_CallbackSet ( spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_SPI_Close()

fsp_err_t R_SCI_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

**Parameters**

| | |
|---|---|
| p_api_ctrl | Pointer to an opened instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SCI_SPI_CalculateBitrate()

fsp_err_t R_SCI_SPI_CalculateBitrate ( uint32_t *bitrate*, sci_spi_div_setting_t * *sclk_div*, bool *use_mddr* )

Calculate the register settings required to achieve the desired bitrate.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc. |
| | sclk_div | Pointer to sci_spi_div_setting_t used to configure baudrate settings. |
| [in] | use_mddr | Calculate the divider settings for use with MDDR. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully. |
| FSP_ERR_ASSERTION | Baud rate is not achievable. |

***Note***

*The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.*

## 5.2.6.25 SPI (r_spi)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SPI_Close (spi_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | R_SPI_CalculateBitrate (uint32_t bitrate, rspck_div_setting_t *spck_div) |

### Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data is sampled at an odd edge and changes at an even edge.
    - CPHA=1 Data changes at an odd edge and is sampled at an even edge.
  - MSB/LSB first
  - 8-16 bit, 20-bit, 24-bit, and 32-bit data frames
    - Hardware endian swap in 16-bit and 32-bit modes
  - SSL level keep (burst transfer) supported if available
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- DMAC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)
  - TX Underrun Error (No data to load into shift register for transmitting)
  - Parity Error (When parity is enabled and a parity error is detected)

# Configuration

### Build Time Configurations for r_spi

The following build time configurations are defined in fsp_cfg/r_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable Support for using a transfer API | • Enabled<br>• Disabled | Enabled | If enabled, transfer instances will be included in the build for |

| | | | |
|---|---|---|---|
| | | | both transmission and reception. |
| Enable Transmitting from RXI Interrupt | • Enabled<br>• Disabled | Disabled | If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a performance boost when neither DTC nor DMAC is used. In addition, Transmit Only mode is not supported when this configuration is enabled. |

**Configurations for Connectivity > SPI (r_spi)**

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SPI channel. |
| Receive Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Transmit Buffer Empty Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Transfer Complete Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Error Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |

| | | | |
|---|---|---|---|
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |
| Callback | Name must be a valid C symbol | spi_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| SPI Mode | • SPI Operation<br>• Clock Synchronous Operation | Clock Synchronous Operation | Select the clock sync mode. |
| Full or Transmit Only Mode | • Full Duplex<br>• Transmit Only | Full Duplex | Select Full Duplex or Transmit Only Mode. |
| Slave Select Polarity | • Active Low<br>• Active High | Active Low | Select the slave select active level. |
| Select SSL(Slave Select) | MCU Specific Options | | Select which slave to use. |
| MOSI Idle State | • MOSI Idle Value Fixing Disable<br>• MOSI Idle Value Fixing Low<br>• MOSI Idle Value Fixing High | MOSI Idle Value Fixing Disable | Select the MOSI idle level if MOSI idle is enabled. |
| Parity Mode | • Disabled<br>• Odd<br>• Even | Disabled | Select the parity mode if parity is enabled. |
| Byte Swapping | • Disable<br>• Enable | Disable | Select the byte swap mode for 16/32-Bit Data Frames. |
| Bitrate | Value must be an integer greater than 0 | 16000000 | Enter the desired bitrate, change the bitrate to a value supported by MCU. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The actual bitrate is printed |

| | | | |
|---|---|---|---|
| | | | in a comment in the generated spi_extended_cfg_t structure. |
| Clock Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles before each data frame. |
| SSL Negation Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles after each data frame. |
| Next Access Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles between each data frame. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA2A2 | PCLKB |
| RA2E1 | PCLKB |
| RA2E2 | PCLKB |
| RA2E3 | PCLKB |
| RA2L1 | PCLKB |
| RA2L2 | PCLKB |
| RA2T1 | PCLKB |
| RA4C1 | PCLKA |
| RA4E1 | PCLKA |
| RA4E2 | PCLKA |

| | |
|---|---|
| RA4L1 | PCLKA |
| RA4M1 | PCLKA |
| RA4M2 | PCLKA |
| RA4M3 | PCLKA |
| RA4T1 | PCLKA |
| RA4W1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T1 | PCLKA |
| RA6T3 | PCLKA |

### Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

*Note*

*It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.*

# Usage Notes

### Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC or DMAC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when possible. A typical use-case where this is possible is when reading/writing to a block device.

### Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full isr. This only improves performance when neither the DTC nor DMAC is being used.

*Note*

*Configuring the module to use an RX DTC/DMAC instance without also providing a TX DTC/DMAC instance results in an invalid configuration when RXI transmit is enabled.*
*Transmit Only mode is not supported when Transmit from RXI is enabled.*

### Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

### Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

### Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- In order to use CPHA=0 setting in slave mode, the master must toggle the SSL pin after every data frame (Even if the device supports SSL Level Keeping). Because of this hardware limitation, the module will return FSP_ERR_UNSUPPORTED when it is configured to use CPHA=0 setting in slave mode.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.
- The behavior for Byte Swap operation is not guaranteed for data frames other than 8-bit, 16-bit and 32bit.

# Examples

### Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;

void spi_basic_example (void)

{

    uint8_t tx_buffer[TRANSFER_SIZE];

    uint8_t rx_buffer[TRANSFER_SIZE];

 fsp_err_t err = FSP_SUCCESS;
```

```
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start a write/read transfer */
    err = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
```

### Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
```

```
/* Initialize the SPI module. */
   err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Assert Slave Select Line 1 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
/* Start a write/read transfer */
   g_transfer_complete = false;
   err                 = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
   assert(FSP_SUCCESS == err);
/* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
while (false == g_transfer_complete)
   {
      ;
   }
/* De-assert Slave Select Line 1 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
/* Wait for minimum time required between transfers. */
R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
/* Assert Slave Select Line 2 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
/* Start a write/read transfer */
   g_transfer_complete = false;
   err                 = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
   assert(FSP_SUCCESS == err);
/* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
while (false == g_transfer_complete)
   {
      ;
   }
/* De-assert Slave Select Line 2 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
```

```
}
```

### Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
    err = R_SPI_CalculateBitrate(BITRATE, &(g_spi_extended_cfg.spck_div));
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | rspck_div_setting_t |
| struct | spi_extended_cfg_t |
| struct | spi_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_ssl_mode_t |
| enum | spi_communication_t |
| enum | spi_ssl_polarity_t |
| enum | spi_ssl_select_t |
| enum | spi_mosi_idle_value_fixing_t |
| enum | spi_parity_t |
| enum | spi_byte_swap_t |

| | |
|---|---|
| enum | spi_delay_count_t |

## Data Structure Documentation

### ◆ rspck_div_setting_t

| struct rspck_div_setting_t | | |
|---|---|---|
| SPI Clock Divider settings. | | |
| Data Fields | | |
| uint8_t | spbr | SPBR register setting. |
| uint8_t | brdv: 2 | BRDV setting in SPCMD0. |

### ◆ spi_extended_cfg_t

| struct spi_extended_cfg_t | | |
|---|---|---|
| Extended SPI interface configuration | | |
| Data Fields | | |
| spi_ssl_mode_t | spi_clksyn | Select spi or clock syn mode operation. |
| spi_communication_t | spi_comm | Select full-duplex or transmit-only communication. |
| spi_ssl_polarity_t | ssl_polarity | Select SSLn signal polarity. |
| spi_ssl_select_t | ssl_select | Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3. |
| spi_mosi_idle_value_fixing_t | mosi_idle | Select MOSI idle fixed value and selection. |
| spi_parity_t | parity | Select parity and enable/disable parity. |
| spi_byte_swap_t | byte_swap | Select byte swap mode. |
| rspck_div_setting_t | spck_div | Register values for configuring the SPI Clock Divider. |
| spi_delay_count_t | spck_delay | SPI Clock Delay Register Setting. |
| spi_delay_count_t | ssl_negation_delay | SPI Slave Select Negation Delay Register Setting. |
| spi_delay_count_t | next_access_delay | SPI Next-Access Delay Register Setting. |

### ◆ spi_instance_ctrl_t

| struct spi_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called. |

| Data Fields | |
|---|---|
| uint32_t | open |
| | Indicates whether the open() API has been successfully called. |
| | |
| spi_cfg_t const * | p_cfg |
| | Pointer to instance configuration. |
| | |
| R_SPI0_Type * | p_regs |
| | Base register for this channel. |
| | |
| void const * | p_tx_data |
| | Buffer to transmit. |
| | |
| void * | p_rx_data |
| | Buffer to receive. |
| | |
| uint32_t | tx_count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| | |
| uint32_t | rx_count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| | |
| uint32_t | count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| | |
| spi_bit_width_t | bit_width |
| | Bits per Data frame (8-bit, 16-bit, 32-bit) |
| | |

## Enumeration Type Documentation

### ◆ spi_ssl_mode_t

| enum spi_ssl_mode_t | |
|---|---|
| 3-Wire or 4-Wire mode. | |
| Enumerator | |
| SPI_SSL_MODE_SPI | SPI operation (4-wire method) |
| SPI_SSL_MODE_CLK_SYN | Clock Synchronous operation (3-wire method) |

### ◆ spi_communication_t

| enum spi_communication_t | |
|---|---|
| Transmit Only (Half Duplex), or Full Duplex. | |
| Enumerator | |
| SPI_COMMUNICATION_FULL_DUPLEX | Full-Duplex synchronous serial communication. |
| SPI_COMMUNICATION_TRANSMIT_ONLY | Transit only serial communication. |

### ◆ spi_ssl_polarity_t

| enum spi_ssl_polarity_t | |
|---|---|
| Slave Select Polarity. | |
| Enumerator | |
| SPI_SSLP_LOW | SSLP signal polarity active low. |
| SPI_SSLP_HIGH | SSLP signal polarity active high. |

### ◆ spi_ssl_select_t

| enum spi_ssl_select_t | |
|---|---|
| The Slave Select Line | |
| Enumerator | |
| SPI_SSL_SELECT_SSL0 | Select SSL0. |
| SPI_SSL_SELECT_SSL1 | Select SSL1. |
| SPI_SSL_SELECT_SSL2 | Select SSL2. |
| SPI_SSL_SELECT_SSL3 | Select SSL3. |

### ◆ spi_mosi_idle_value_fixing_t

| enum spi_mosi_idle_value_fixing_t | |
|---|---|
| MOSI Idle Behavior. | |
| Enumerator | |
| SPI_MOSI_IDLE_VALUE_FIXING_DISABLE | MOSI output value=value set in MOIFV bit. |
| SPI_MOSI_IDLE_VALUE_FIXING_LOW | MOSIn level low during MOSI idling. |
| SPI_MOSI_IDLE_VALUE_FIXING_HIGH | MOSIn level high during MOSI idling. |

### ◆ spi_parity_t

| enum spi_parity_t | |
|---|---|
| Parity Mode | |
| Enumerator | |
| SPI_PARITY_MODE_DISABLE | Disable parity. |
| SPI_PARITY_MODE_ODD | Select even parity. |
| SPI_PARITY_MODE_EVEN | Select odd parity. |

### ◆ spi_byte_swap_t

| enum spi_byte_swap_t | |
|---|---|
| Byte Swapping Enable/Disable. | |
| Enumerator | |
| SPI_BYTE_SWAP_DISABLE | Disable Byte swapping for 16/32-Bit transfers. |
| SPI_BYTE_SWAP_ENABLE | Enable Byte swapping for 16/32-Bit transfers. |

### ◆ spi_delay_count_t

| enum spi_delay_count_t | |
|---|---|
| Delay count for SPI delay settings. | |
| Enumerator | |
| SPI_DELAY_COUNT_1 | Set RSPCK delay count to 1 RSPCK. |
| SPI_DELAY_COUNT_2 | Set RSPCK delay count to 2 RSPCK. |
| SPI_DELAY_COUNT_3 | Set RSPCK delay count to 3 RSPCK. |
| SPI_DELAY_COUNT_4 | Set RSPCK delay count to 4 RSPCK. |
| SPI_DELAY_COUNT_5 | Set RSPCK delay count to 5 RSPCK. |
| SPI_DELAY_COUNT_6 | Set RSPCK delay count to 6 RSPCK. |
| SPI_DELAY_COUNT_7 | Set RSPCK delay count to 7 RSPCK. |
| SPI_DELAY_COUNT_8 | Set RSPCK delay count to 8 RSPCK. |

## Function Documentation

### ◆ R_SPI_Open()

fsp_err_t R_SPI_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the pperipheral registers according to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ALREADY_OPEN | Instance was already initialized. |
| FSP_ERR_ASSERTION | An invalid argument was given in the configuration structure. |
| FSP_ERR_UNSUPPORTED | A requested setting is not possible on this device with the current build configuration. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

***Note***

*This function is reentrant.*

### ◆ R_SPI_Read()

fsp_err_t R_SPI_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_Write()

fsp_err_t R_SPI_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or source parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_WriteRead()

fsp_err_t R_SPI_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control, source or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_CallbackSet()

fsp_err_t R_SPI_CallbackSet ( spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SPI_Close()

fsp_err_t R_SPI_Close ( spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | A required pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SPI_CalculateBitrate()

fsp_err_t R_SPI_CalculateBitrate ( uint32_t *bitrate*, rspck_div_setting_t * *spck_div* )

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | Desired bitrate |
| [out] | spck_div | Memory location to store bitrate register settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Valid spbr and brdv values were calculated |
| FSP_ERR_UNSUPPORTED | Bitrate is not achievable |

## 5.2.6.26 SPI (r_spi_b)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SPI_B_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg) |
| fsp_err_t | R_SPI_B_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_B_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_B_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| fsp_err_t | R_SPI_B_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SPI_B_Close (spi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SPI_B_CalculateBitrate (uint32_t bitrate, spi_b_clock_source_t clock_source, rspck_div_setting_t *spck_div) |

### Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.

# Overview

### Features

- Standard SPI Modes
  - Master or Slave Mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data is sampled at an odd edge and changes at an even edge.
    - CPHA=1 Data changes at an odd edge and is sampled at an even edge.
  - MSB/LSB first
  - 8- to 32-Bit data frames
    - Hardware endian swap in 16-Bit and 32-Bit mode
  - SSL level keep (burst transfer) supported
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- DMAC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)
  - TX Underrun Error (No data to load into shift register for transmitting)
  - Parity Error (When parity is enabled and a parity error is detected)

# Configuration

### Build Time Configurations for r_spi_b

The following build time configurations are defined in fsp_cfg/r_spi_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable Support for using a transfer API | • Enabled<br>• Disabled | Enabled | If enabled, transfer instances will be included in the build for both transmission and reception. |
| Enable Transmitting from RXI Interrupt | • Enabled<br>• Disabled | Disabled | If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a |

| | | | |
|---|---|---|---|
| | | | performance boost when DTC is not used. In addition, Transmit Only mode is not supported when this configuration is enabled. |

**Configurations for Connectivity > SPI (r_spi_b)**

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_spi_b). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_spi0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SPI channel. |
| Receive Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Transmit Buffer Empty Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Transfer Complete Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Error Interrupt Priority | MCU Specific Options | | Select the interrupt priority for all SPI interrupts. |
| Operating Mode | • Master<br>• Slave | Master | Select the SPI operating mode. |
| Clock Phase | • Data sampling on odd edge, data variation on even edge<br>• Data sampling on even edge, data variation on odd edge | Data sampling on odd edge, data variation on even edge | Select the clock edge to sample data. |
| Clock Polarity | • Low when idle<br>• High when idle | Low when idle | Select clock level when idle. |
| Mode Fault Error | • Enable<br>• Disable | Disable | Detect master/slave mode conflicts. |
| Bit Order | • MSB First<br>• LSB First | MSB First | Select the data bit order. |

| | | | |
|---|---|---|---|
| Callback | Name must be a valid C symbol | spi_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| SPI Mode | • SPI Operation<br>• Clock Synchronous Operation | Clock Synchronous Operation | Select the clock sync mode. |
| Communication Mode Select | • Full Duplex<br>• Transmit Only | Full Duplex | Select Full Duplex or Transmit Only Mode. |
| Slave Select Polarity | • Active Low<br>• Active High | Active Low | Select the slave select active level. |
| Select SSL(Slave Select) | • SSL0<br>• SSL1<br>• SSL2<br>• SSL3 | SSL0 | Select which slave to use. |
| MOSI Idle State | • MOSI Idle Value Fixing Disable<br>• MOSI Idle Value Fixing Low<br>• MOSI Idle Value Fixing High | MOSI Idle Value Fixing Disable | Select the MOSI idle level if MOSI idle is enabled. |
| Parity Mode | • Disabled<br>• Odd<br>• Even | Disabled | Select the parity mode if parity is enabled. |
| Byte Swapping | • Disable<br>• Enable | Disable | Select the byte swap mode for 16/32-Bit Data Frames. |
| Clock Source | MCU Specific Options | | Select the clock source for communication. |
| Bitrate | Value must be an integer greater than 0 | 16000000 | Enter the desired bitrate, change the bitrate to a value supported by MCU. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The actual bitrate is printed in a comment in the generated spi_extended_cfg_t |

| | | | |
|---|---|---|---|
| | | | structure. |
| Clock Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles before each data frame. |
| SSL Negation Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles after each data frame. |
| Next Access Delay | • 1 Clock<br>• 2 Clocks<br>• 3 Clocks<br>• 4 Clocks<br>• 5 Clocks<br>• 6 Clocks<br>• 7 Clocks<br>• 8 Clocks | 1 Clock | Configure the number of SPI clock cycles between each data frame. |

### Clock Configuration

The SPI peripheral uses the SCISPICLK for communication and PCLKB for internal operations. Both can be configured via the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

*Note*

*It is recommended to configure the pins with high drive strength (or high-speed high if available) to maintain signal integrity, particularly at high bitrates. The drive strength can be set by opening the e2 studio pin configuration editor, selecting the pin, and configuring 'Drive Capacity'.*

# Usage Notes

### Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC or DMAC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when

possible. A typical use-case where this is possible is when reading/writing to a block device.

### Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full ISR. This only improves performance when the DTC nor DMAC is not being used.

*Note*

*Configuring the module to use RX DTC/DMAC instance without also providing a TX DTC/DMAC instance results in an invalid configuration when RXI transmit is enabled.*
*Transmit Only mode is not supported when Transmit from RXI is enabled.*

### Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

### Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

### Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.
- The behavior for Byte Swap operation is not guaranteed for data frames other than 8-bit, 16-bit and 32bit.
- Full-duplex transmission and reception (writeRead) may experience RX overflows at very high bitrates when using DTC or DMAC. In this case, use half-duplex (write and read) functions.

## Examples

### Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;
```

```
void spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start a write/read transfer */
    err = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
```

### Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
```

```
/* Configure Slave Select Line 1 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
/* Configure Slave Select Line 2 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
fsp_err_t err = FSP_SUCCESS;
/* Initialize the SPI module. */
   err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Assert Slave Select Line 1 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
/* Start a write/read transfer */
   g_transfer_complete = false;
   err                 = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
   assert(FSP_SUCCESS == err);
/* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
while (false == g_transfer_complete)
   {
      ;
   }
/* De-assert Slave Select Line 1 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
/* Wait for minimum time required between transfers. */
R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
/* Assert Slave Select Line 2 */
R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
/* Start a write/read transfer */
   g_transfer_complete = false;
   err                 = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
   assert(FSP_SUCCESS == err);
/* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
while (false == g_transfer_complete)
```

```
    {
        ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

### Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
    err = R_SPI_B_CalculateBitrate(BITRATE, SPI_B_CLOCK_SOURCE_SCISPICLK,
&(g_spi_extended_cfg.spck_div));
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | rspck_div_setting_t |
| struct | spi_b_extended_cfg_t |
| struct | spi_b_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_b_ssl_mode_t |
| enum | spi_b_communication_t |
| enum | spi_b_ssl_polarity_t |

| | |
|---|---|
| enum | spi_b_ssl_select_t |
| enum | spi_b_mosi_idle_value_fixing_t |
| enum | spi_b_parity_t |
| enum | spi_b_byte_swap_t |
| enum | spi_b_delay_count_t |
| enum | spi_b_clock_source_t |

## Data Structure Documentation

### ◆ rspck_div_setting_t

| struct rspck_div_setting_t | | |
|---|---|---|
| SPI Clock Divider settings. | | |
| Data Fields | | |
| uint8_t | spbr | SPBR register setting. |
| uint8_t | brdv: 2 | BRDV setting in SPCMD0. |

### ◆ spi_b_extended_cfg_t

| struct spi_b_extended_cfg_t | | |
|---|---|---|
| Extended SPI interface configuration | | |
| Data Fields | | |
| spi_b_ssl_mode_t | spi_clksyn | Select SPI or Clock Synchronous mode operation. |
| spi_b_communication_t | spi_comm | Select full-duplex or transmit-only communication. |
| spi_b_ssl_polarity_t | ssl_polarity | Select SSLn signal polarity. |
| spi_b_ssl_select_t | ssl_select | Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3. |
| spi_b_mosi_idle_value_fixing_t | mosi_idle | Select MOSI idle fixed value and selection. |
| spi_b_parity_t | parity | Select parity and enable/disable parity. |
| spi_b_byte_swap_t | byte_swap | Select byte swap mode. |
| spi_b_clock_source_t | clock_source | Communication clock source (TCLK). |
| rspck_div_setting_t | spck_div | Register values for configuring the SPI Clock Divider. |

| | | |
|---|---|---|
| spi_b_delay_count_t | spck_delay | SPI Clock Delay Register Setting. |
| spi_b_delay_count_t | ssl_negation_delay | SPI Slave Select Negation Delay Register Setting. |
| spi_b_delay_count_t | next_access_delay | SPI Next-Access Delay Register Setting. |

### ◆ spi_b_instance_ctrl_t

| struct spi_b_instance_ctrl_t | |
|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Indicates whether the open() API has been successfully called. |
| | |
| spi_cfg_t const * | p_cfg |
| | Pointer to instance configuration. |
| | |
| R_SPI_B0_Type * | p_regs |
| | Base register for this channel. |
| | |
| void const * | p_tx_data |
| | Buffer to transmit. |
| | |
| void * | p_rx_data |
| | Buffer to receive. |
| | |
| uint32_t | tx_count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| | |
| uint32_t | rx_count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |

| | |
|---|---|
| | |
| uint32_t | count |
| | Number of Data Frames to transfer (8-bit, 16-bit, 32-bit) |
| | |
| spi_bit_width_t | bit_width |
| | Bits per Data frame (8-bit, 16-bit, 32-bit) |
| | |

### Enumeration Type Documentation

#### ◆ spi_b_ssl_mode_t

| enum spi_b_ssl_mode_t | |
|---|---|
| 3-Wire or 4-Wire mode. | |
| Enumerator | |
| SPI_B_SSL_MODE_SPI | SPI operation (4-wire method) |
| SPI_B_SSL_MODE_CLK_SYN | Clock Synchronous operation (3-wire method) |

#### ◆ spi_b_communication_t

| enum spi_b_communication_t | |
|---|---|
| Transmit Only (Half Duplex), or Full Duplex. | |
| Enumerator | |
| SPI_B_COMMUNICATION_FULL_DUPLEX | Full-Duplex synchronous serial communication. |
| SPI_B_COMMUNICATION_TRANSMIT_ONLY | Transit only serial communication. |

### ◆ spi_b_ssl_polarity_t

| enum spi_b_ssl_polarity_t | |
|---|---|
| Slave Select Polarity. | |
| Enumerator | |
| SPI_B_SSLP_LOW | SSLP signal polarity active low. |
| SPI_B_SSLP_HIGH | SSLP signal polarity active high. |

### ◆ spi_b_ssl_select_t

| enum spi_b_ssl_select_t | |
|---|---|
| The Slave Select Line | |
| Enumerator | |
| SPI_B_SSL_SELECT_SSL0 | Select SSL0. |
| SPI_B_SSL_SELECT_SSL1 | Select SSL1. |
| SPI_B_SSL_SELECT_SSL2 | Select SSL2. |
| SPI_B_SSL_SELECT_SSL3 | Select SSL3. |

### ◆ spi_b_mosi_idle_value_fixing_t

| enum spi_b_mosi_idle_value_fixing_t | |
|---|---|
| MOSI Idle Behavior. | |
| Enumerator | |
| SPI_B_MOSI_IDLE_VALUE_FIXING_DISABLE | MOSI output value=value set in MOIFV bit. |
| SPI_B_MOSI_IDLE_VALUE_FIXING_LOW | MOSIn level low during MOSI idling. |
| SPI_B_MOSI_IDLE_VALUE_FIXING_HIGH | MOSIn level high during MOSI idling. |

### ◆ spi_b_parity_t

| enum spi_b_parity_t | |
|---|---|
| Parity Mode | |
| Enumerator | |
| SPI_B_PARITY_MODE_DISABLE | Disable parity. |
| SPI_B_PARITY_MODE_ODD | Select even parity. |
| SPI_B_PARITY_MODE_EVEN | Select odd parity. |

### ◆ spi_b_byte_swap_t

| enum spi_b_byte_swap_t | |
|---|---|
| Byte Swapping Enable/Disable. | |
| Enumerator | |
| SPI_B_BYTE_SWAP_DISABLE | Disable Byte swapping for 16/32-Bit transfers. |
| SPI_B_BYTE_SWAP_ENABLE | Enable Byte swapping for 16/32-Bit transfers. |

### ◆ spi_b_delay_count_t

| enum spi_b_delay_count_t | |
|---|---|
| Delay count for SPI delay settings. | |
| Enumerator | |
| SPI_B_DELAY_COUNT_1 | Set RSPCK delay count to 1 RSPCK. |
| SPI_B_DELAY_COUNT_2 | Set RSPCK delay count to 2 RSPCK. |
| SPI_B_DELAY_COUNT_3 | Set RSPCK delay count to 3 RSPCK. |
| SPI_B_DELAY_COUNT_4 | Set RSPCK delay count to 4 RSPCK. |
| SPI_B_DELAY_COUNT_5 | Set RSPCK delay count to 5 RSPCK. |
| SPI_B_DELAY_COUNT_6 | Set RSPCK delay count to 6 RSPCK. |
| SPI_B_DELAY_COUNT_7 | Set RSPCK delay count to 7 RSPCK. |
| SPI_B_DELAY_COUNT_8 | Set RSPCK delay count to 8 RSPCK. |

### ◆ spi_b_clock_source_t

| enum spi_b_clock_source_t |
|---|
| SPI communication clock source. |

## Function Documentation

### ◆ R_SPI_B_Open()

fsp_err_t R_SPI_B_Open ( spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg* )

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the peripheral registers according to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel initialized successfully. |
| FSP_ERR_ALREADY_OPEN | Instance was already initialized. |
| FSP_ERR_ASSERTION | An invalid argument was given in the configuration structure. |
| FSP_ERR_UNSUPPORTED | A requested setting is not possible on this device with the current build configuration. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel number is invalid. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

***Note***

*This function is reentrant.*

### ◆ R_SPI_B_Read()

fsp_err_t R_SPI_B_Read ( spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Read operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_B_Write()

fsp_err_t R_SPI_B_Write ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control or source parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_B_WriteRead()

fsp_err_t R_SPI_B_WriteRead ( spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width* )

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write operation successfully completed. |
| FSP_ERR_ASSERTION | NULL pointer to control, source or destination parameters or transfer length is zero. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |
| FSP_ERR_IN_USE | A transfer is already in progress. |

### ◆ R_SPI_B_CallbackSet()

fsp_err_t R_SPI_B_CallbackSet ( spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void *const *p_context*, spi_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SPI_B_Close()

fsp_err_t R_SPI_B_Close ( spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | A required pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The channel has not been opened. Open the channel first. |

### ◆ R_SPI_B_CalculateBitrate()

fsp_err_t R_SPI_B_CalculateBitrate ( uint32_t *bitrate*, spi_b_clock_source_t *clock_source*, rspck_div_setting_t * *spck_div* )

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

**Parameters**

| | | |
|---|---|---|
| [in] | bitrate | Desired bitrate |
| [in] | clock_source | SPI communication clock source to be used |
| [out] | spck_div | Memory location to store bitrate register settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Valid spbr and brdv values were calculated |
| FSP_ERR_UNSUPPORTED | Bitrate is not achievable |

## 5.2.6.27 SPI Communication Device (rm_comms_spi)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | RM_COMMS_SPI_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg)<br>Opens and configures the SPI Comms module. Implements rm_comms_api_t::open. More... |
| fsp_err_t | RM_COMMS_SPI_Close (rm_comms_ctrl_t *const p_api_ctrl)<br>Disables specified SPI Comms module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_SPI_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)<br>Updates the SPI Comms callback. Implements rm_comms_api_t::callbackSet. More... |

| | |
|---|---|
| fsp_err_t | RM_COMMS_SPI_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)<br>Performs a read from the SPI bus. Implements rm_comms_api_t::read . More... |
| fsp_err_t | RM_COMMS_SPI_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write to the SPI bus. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_SPI_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs simultaneous read and write. Implements rm_comms_api_t::writeRead.. More... |
| void | rm_comms_spi_callback (spi_callback_args_t *p_args)<br>Common callback function called in the SPI driver callback function. |

### Detailed Description

Middleware to implement a generic communications interface over SPI. This module implements the Communications Middleware Interface.

# Overview

The RM_COMMS_SPI module implements COMMS API for SPI interface.

### Features

The implementation of the SPI communications interface has the following key features:

- Read data from SPI bus
- Write data to SPI bus
- Simultaneously read from and write to SPI bus
- Support slave select and multiple slaves

# Configuration

### Build Time Configurations for rm_comms_spi

The following build time configurations are defined in fsp_cfg/rm_comms_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP) | Default (BSP) | If selected code for |

| | | | |
|---|---|---|---|
| | • Enabled<br>• Disabled | | parameter checking is included in the build. |
| Auto Slave Select | • Enabled<br>• Disabled | Disabled | if selected slave select pin will be configured by module instance. |

**Configurations for Connectivity > SPI Shared Bus (rm_comms_spi)**

This module can be added to the Stacks tab via New Stack > Connectivity > SPI Shared Bus (rm_comms_spi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comms_spi_bus0 | Module name. |
| Bus Timeout | Value must be a non-negative integer less than or equal to 0xFFFFFFFF | 0xFFFFFFFF | Set timeout for locking bus in using RTOS. |
| Semaphore for Blocking (RTOS only) | • Unuse<br>• Use | Use | Set Semaphore for blocking in using RTOS. |
| Recursive Mutex for Bus (RTOS only) | • Unuse<br>• Use | Use | Set Mutex for locking bus in using RTOS. |

**Configurations for Connectivity > SPI Communication Device (rm_comms_spi)**

This module can be added to the Stacks tab via New Stack > Connectivity > SPI Communication Device (rm_comms_spi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_comms_spi_device0 | Module name. |
| SSL Port Number | Refer to the RA Configuration tool for available options. | Disabled | GPIO output port number to select slave device. |
| SSL Pin Number | Refer to the RA Configuration tool for available options. | Disabled | GPIO output pin number to select slave device. |
| SSL Next Access Delay (Microseconds) | Value must be a non-negative integer | 1 | SSL next access delay. |
| Slave Select Polarity | • Active Low<br>• Active High | Active Low | Select the slave select active level. |
| Semaphore Timeout (RTOS only) | Value must be a non-negative integer less than or equal to 0xFFFFFFFF | 0xFFFFFFFF | Set timeout for blocking in using RTOS. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be |

provided.

### Pin Configuration

A GPIO can be configured for slave select.

## Usage Notes

If an RTOS is used, blocking and bus lock is available via SPI shared bus. Bus lock is only available when blocking is used.

### Limitations

- Only 8-bit data width is supported
- RM_COMMS_API are not reentrant in non blocking mode
- When in blocking mode, RM_COMMS_SPI_Write(), RM_COMMS_SPI_Read() and RM_COMMS_SPI_WriteRead() cannot be called in callback.
- COMMS SPI devices on the same SPI bus must sit on the same SPI shared bus.

## Examples

### Basic Example

This is a basic example of minimal use of SPI communications implementation in an application.

```
void rm_comms_spi_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_COMMS_SPI_Open(&g_comms_spi_ctrl, &g_comms_spi_cfg);
 if (FSP_SUCCESS != err)
    {
 /* Handle any errors. */
    }
 while (true)
    {
 /* Send data. */
       g_error    = 0;
       g_completed = 0;
       err = RM_COMMS_SPI_Write(&g_comms_spi_ctrl, g_tx_buf, TX_BUF_LEN);
 if (FSP_SUCCESS != err)
     {
 /* Handle any errors. */
     }
```

```
 while ((0 == g_completed) && (0 == g_error))
      {
 /* Wait callback */
      }
 /* Receive data. */
       g_error    = 0;
       g_completed = 0;
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
       err = RM_COMMS_SPI_Read(&g_comms_spi_ctrl, g_rx_buf, RX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors.*/
      }
 while ((0 == g_completed) && (0 == g_error))
      {
 /* Wait callback */
      }
    }
}
static void rm_comms_user_callback (rm_comms_callback_args_t * p_args)
{
 if (p_args->event == RM_COMMS_EVENT_OPERATION_COMPLETE)
    {
       g_completed = 1;
    }
 else
    {
       g_error = 1;
    }
}
```

### Data Structures

| | |
|---|---|
| struct | rm_comms_spi_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_comms_spi_instance_ctrl_t

| struct rm_comms_spi_instance_ctrl_t | |
|---|---|
| Communications middleware control structure. | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_comms_cfg_t const * | p_cfg |
| | Middleware configuration. |
| | |
| rm_comms_spi_bus_extended_cfg_t * | p_bus |
| | Pointer to extended configuration structure. |
| | |
| rm_comms_spi_device_extended_cfg_t * | p_device |
| | Pointer to device specific settings (lower level configuration structure) |
| | |
| void(* | p_callback )(rm_comms_callback_args_t *p_args) |
| | Pointer to callback that is called when a spi_event_t occurs. |
| | |
| void * | p_context |
| | Pointer to context passed into callback function. |
| | |

## Function Documentation

### ◆ RM_COMMS_SPI_Open()

fsp_err_t RM_COMMS_SPI_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the SPI Comms module. Implements rm_comms_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI Comms module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SPI_Close()

fsp_err_t RM_COMMS_SPI_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified SPI Comms module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SPI_CallbackSet()

fsp_err_t RM_COMMS_SPI_CallbackSet ( rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context* )

Updates the SPI Comms callback. Implements rm_comms_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_SPI_Read()

fsp_err_t RM_COMMS_SPI_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the SPI bus. Implements rm_comms_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SPI_Write()

fsp_err_t RM_COMMS_SPI_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the SPI bus. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SPI_WriteRead()

fsp_err_t RM_COMMS_SPI_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs simultaneous read and write. Implements rm_comms_api_t::writeRead..

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer was started successfully. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Read bytes and write bytes are not the same. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.6.28 UART (r_sau_uart)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SAU_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) |
| fsp_err_t | R_SAU_UART_Close (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SAU_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SAU_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_SAU_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SAU_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info) |
| fsp_err_t | R_SAU_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort) |

| | |
|---|---|
| fsp_err_t | R_SAU_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes) |
| fsp_err_t | R_SAU_UART_BaudCalculate (sau_uart_instance_ctrl_t *const p_ctrl, uint32_t baudrate, sau_uart_baudrate_setting_t *const p_baud_setting) |
| fsp_err_t | R_SAU_UART_ReceiveSuspend (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SAU_UART_ReceiveResume (uart_ctrl_t *const p_api_ctrl) |

### Detailed Description

UART driver for the SAU peripheral on RA MCUs. This module implements the UART Interface.

# Overview

### Features

The SAU UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Integration with the DTC transfer module
- Abort in-progress read/write operations

# Configuration

### Build Time Configurations for r_sau_uart

The following build time configurations are defined in fsp_cfg/r_sau_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Critical Section Guarding | • Enabled<br>• Disabled | Disabled | Enable critical section guarding around peripheral configuration updates. This should be enabled if the R_SAU_I2C module is being used simultaneously with |

| | | | |
|---|---|---|---|
| | | | this module. |
| DTC Support | • Enable<br>• Disable | Disable | Enable DTC support for the SAU_UART module. |
| Enable Single Channel | MCU Specific Options | | Enable single channel to reduce code size if only channel 0, 1, or 2 is needed. |
| Enable Fixed Baudrate | • Enable<br>• Disable | Enable | Disable baudrate calculation and setter functions to reduce code size. |

**Configurations for Connectivity > UART (r_sau_uart)**

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sau_uart).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_uart0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the UART channel. |
| Data Bits | • 7 bits<br>• 8 bits<br>• 9 bits | 8 bits | Select the number of bits per word. |
| Parity | • None<br>• Zero<br>• Odd<br>• Even | None | Select the parity mode. |
| Stop Bits | • 1 bit<br>• 2 bits | 1 bit | Select the number of stop bits. In receive, 2 bit is not available. |
| Bit Order | • LSB First<br>• MSB First | LSB First | Select of data transfer sequence. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 115200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, adjust the operation clock frequency until the baud rate is achievable. The calculated baud rate is printed in a comment |

| | | | |
|---|---|---|---|
| | | | in the generated sau_uart_baudrate_setting_t structure. |
| Extra | | | |
| Operation Clock | • CKm0<br>• CKm1 | CKm0 | Select the operation clock. Use the Clocks tab to set the operation clock divider. |
| Tx Signal Level | • Standard<br>• Inverted | Standard | Select the level of transmitted signal. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Receive End Interrupt Priority | MCU Specific Options | | Select the receive end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |

### Clock Configuration

The SAU clock uses the system clock (ICLK) as its clock source.

A prescaler is applied to the ICLK in order to produce the operation clock frequency. The operation clock is used to generate the desired transfer period of the SAU module.

SAU operation clocks are shared among all channels within a SAU unit. Check the Hardware User's Manual for your MCU for available units and channels. SAU operation clock dividers are configurable in the Clocks tab.

The operation clock dividers are named SAU CKmn where m is the SAU unit, and n is the operation clock. For example, SAU CK01 applies to all SAU0 instances using CK1 as the operation clock (m=0, n=1).

### Pin Configuration

This module uses TXD and RXD to communicate to external devices.

# Usage Notes

### 9-bit data transfers

- p_src: the address for the send data buffer. p_src in R_SAU_UART_Write must be aligned on a 16-bit boundary. (the data order in buffer: the lower 8-bits of the first number, the highest 1-bit of the first number, the lower 8-bits of the second number, the highest 1-bit of the second number, etc)
- byte: the size of send data buffer.

**Selecting Operation Clock Frequency**

The relationship between operation clock frequency and bitrate is: bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ] where:

- SDRmn.STCLK is an integer in the range [2, 127] for SAU UART
- f_mck is the operation clock (SAU CKmn) frequency

By plugging in the minimum and maximum SDRmn.STCLK values, the range of bitrates for a given operation clock frequency can be obtained.

Note that due to STCLK being set as discrete integers, the actual bitrate may not be exact. The actual bitrate and percent errors can be calculated by the formulas:

```
actual_bitrate = f_mck / [ 2 * (SDRmn.STCLK + 1) ]
percent_error = 100 * abs [(actual_bitrate - expected_bitrate) / expected_bitrate]
```

Using the fastest possible operation clock for the desired bitrate will result in the lowest deviation from the requested baud rate. Set the CKmn operation clock divider in the Clocks tab to select the desired operation clock frequency.

**Example of Selecting Operation Clock Frequency**

Suppose an example application with a 32MHz ICLK requires:

- 1 SAU0 UART0 instance running at 1000 bps
- 1 SAU0 UART1 instance running at 115200 bps

The following operation clocks could be used:

```
# Operation clock frequency of 250 kHz with 32 Mhz ICLK (32MHz/128=250 kHz)
bitrate_min = 250e3 / [ 2 * (127 + 1) ] = ~976 bps
bitrate_max = 250e3 / [ 2 * (2 + 1) ] = ~41,666 bps
# Operation clock frequency of 16 MHz with 32 MHz ICLK (32MHz/2=16 MHz)
bitrate_min = 16e6 / [ 2 * (127 + 1) ] = ~62,500 bps
bitrate_max = 16e6 / [ 2 * (2 + 1) ] = ~2,666,666 bps
```

An operation clock of 250 kHz works for UART0 because 1000 is in the range [976, 41.6k]. An operation clock of 16 MHz works for UART1 because 115200 is in the range [62.5k, 2.6M].

Applying the settings:

- Select CK0 for UART0 and CK1 for UART1 using the SAU UART "Extra > Operation Clock" property
- Set CK00 Div to /128 for SAU0, CK0 (UART0) in the Clocks tab
- Set CK01 Div to /2 for for SAU0, CK1 (UART1) in the Clocks tab

### Runtime Baud Rate Change

In order to change the baud rate at runtime, "Common > Enable Fixed Baudrate" must be set to "Disabled" in the module properties.

If changing the baud rate is required at runtime, use a unique operation clock for each SAU instance. This is required because R_SAU_UART_BaudSet may change the operation clock divider. Since the operation clocks are shared between SAU channels on each SAU unit, changing a shared operation clock of one instance will cause an incorrect bitrate to be generated on the other instance(s).

If the system clock frequency is changed at runtime, this setting should be disabled, as the baud rate settings will need to be updated after the system clock change.

### Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.
- If 9-bit data length is specified at R_SAU_UART_Open call, p_src in R_SAU_UART_Write must be aligned on a 16-bit boundary.
- When multiple channels on the same SAU unit need to be used, and one is configured as UART, then critical section property needs to be enabled.

### DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_SAU_UART_Read(). The logic required to combine bytes received through R_SAU_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

# Examples

### SAU UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];

uint8_t g_src[TRANSFER_LENGTH];

uint8_t g_out_of_band_received[TRANSFER_LENGTH];

uint32_t g_transfer_complete = 0;
```

```
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sau_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SAU_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_SAU_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SAU_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
```

```
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                  g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                  uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                  *p_dest             = (uint16_t) p_args->data;
                  g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
    }
}
```

### SAU UART Baud Set Example

```
#define SAU_UART_BAUDRATE_19200 (19200)
#define SAU_UART_BAUDRATE_ERROR_PERCENT_5 (5000)
void r_sau_uart_baud_example (void)
{
    sau_uart_baudrate_setting_t baud_setting;
    uint32_t baud_rate = SAU_UART_BAUDRATE_19200;
 fsp_err_t err = R_SAU_UART_BaudCalculate(&g_uart0_ctrl, baud_rate, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SAU_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | sau_uart_extended_cfg_t |
| struct | sau_uart_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | sau_uart_data_sequence_t |
| enum | sau_operation_clock_t |
| enum | sau_uart_signal_level_t |

## Data Structure Documentation

### ◆ sau_uart_extended_cfg_t

| struct sau_uart_extended_cfg_t | | |
|---|---|---|
| UART Configuration | | |
| Data Fields | | |
| sau_uart_data_sequence_t | sequence | Transfer sequence (LSB or MSB) |
| sau_uart_signal_level_t | signal_level | Transfer data signal level (standard or inverted) |
| sau_uart_baudrate_setting_t * | p_baudrate | Baud rate setting (SPS and SDR value) |

### ◆ sau_uart_instance_ctrl_t

| struct sau_uart_instance_ctrl_t |
|---|
| |

UART instance control block. DO NOT INITIALIZE.

**Data Fields**

| | |
|---|---|
| uint8_t | extra_data_byte |
| | 0 for 7 or 8 bit data length(1-byte), 1 for 9 bit data length(2-byte) |
| | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| uart_cfg_t const * | p_cfg |
| | Pointer to the configuration block. |
| | |
| R_SAU0_Type * | p_reg |
| | Base register for the transmit channel. |
| | |
| uint8_t | sau_unit |
| | SAU unit information. |
| | |
| uint8_t | sau_tx_channel |
| | SAU channel information. |
| | |
| uint8_t * | p_src |
| | Source buffer pointer. |
| | |
| uint32_t | tx_count |
| | Size of destination buffer pointer from transmit ISR. |
| | |
| uint8_t * | p_dest |
| | Destination buffer pointer. |

| | |
|---|---|
| uint32_t | rx_count |
| | Size of destination buffer pointer used for receiving data. |

## Enumeration Type Documentation

### ◆ sau_uart_data_sequence_t

| enum sau_uart_data_sequence_t | |
|---|---|
| UART Data transfer sequence definition | |
| Enumerator | |
| SAU_UART_DATA_SEQUENCE_MSB | Data sequence MSB first. |
| SAU_UART_DATA_SEQUENCE_LSB | Data sequence LSB first. |

### ◆ sau_operation_clock_t

| enum sau_operation_clock_t | |
|---|---|
| UART operation clock selection definition | |
| Enumerator | |
| SAU_UART_OPERATION_CLOCK_CK0 | Operating clock use CK0. |
| SAU_UART_OPERATION_CLOCK_CK1 | Operating clock use CK1. |

### ◆ sau_uart_signal_level_t

| enum sau_uart_signal_level_t | |
|---|---|
| UART data signal level definition | |
| Enumerator | |
| SAU_UART_SIGNAL_LEVEL_STANDARD | Uart data signal level standard. |
| SAU_UART_SIGNAL_LEVEL_INVERTED | Uart data signal level inverted. |

## Function Documentation

### ◆ R_SAU_UART_Open()

fsp_err_t R_SAU_UART_Open ( uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg* )

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Flow control is enabled but flow control pin is not defined or selected channel does not support "Hardware CTS and Hardware RTS" flow control. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SAU_UART_Close()

fsp_err_t R_SAU_UART_Close ( uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SAU_UART_Read()

fsp_err_t R_SAU_UART_Read ( uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Destination address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |
| FSP_ERR_UNSUPPORTED | current operation mode is transmission only. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SAU_UART_Open call, p_dest must be aligned 16-bit boundary.*

### ◆ R_SAU_UART_Write()

fsp_err_t R_SAU_UART_Write ( uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission finished successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Source address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A UART transmission is in progress |
| FSP_ERR_UNSUPPORTED | SAU_UART_CFG_TX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SAU_UART_Open call, p_src must be aligned on a 16-bit boundary.*

### ◆ R_SAU_UART_BaudSet()

fsp_err_t R_SAU_UART_BaudSet ( uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a sau_uart_baudrate_setting_t structure. Implements uart_api_t::baudSet

**Warning**

This terminates any in-progress transmission.
This function may change the operation clock frequency. Select a unique operation clock for each SAU instance if using this function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer p_ctrl is NULL |
| FSP_ERR_INVALID_ARGUMENT | p_api_ctrl is empty. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_UNSUPPORTED | Fixed baud rate is enabled |

### ◆ R_SAU_UART_CallbackSet()

fsp_err_t R_SAU_UART_CallbackSet ( uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void *const *p_context*, uart_callback_args_t *const *p_callback_memory* )

Updates the user callback for callback structure. Implements uart_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer p_ctrl is NULL or p_callback_memory is not NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_SAU_UART_InfoGet()

fsp_err_t R_SAU_UART_InfoGet ( uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info* )

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_INVALID_ARGUMENT | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SAU_UART_Abort()

fsp_err_t R_SAU_UART_Abort ( uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort* )

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SAU_UART_ReadStop()

fsp_err_t R_SAU_UART_ReadStop ( uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes* )

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SAU_UART_BaudCalculate()

fsp_err_t R_SAU_UART_BaudCalculate ( sau_uart_instance_ctrl_t *const *p_ctrl*, uint32_t *baudrate*, sau_uart_baudrate_setting_t *const *p_baud_setting* )

Calculates baud rate register settings (SDR.STCLK) for the specified SAU unit.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SAU UART control block. |
| [in] | baudrate | Baud rate [bps]. For example, 19200, 57600, 115200, etc. |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is successfully calculated |
| FSP_ERR_UNSUPPORTED | Fixed baudrate is being used |
| FSP_ERR_ASSERTION | Null pointer |
| FSP_ERR_INVALID_ARGUMENT | Baud rate is not achievable with selected operation clock frequency |

### ◆ R_SAU_UART_ReceiveSuspend()

fsp_err_t R_SAU_UART_ReceiveSuspend ( uart_ctrl_t *const *p_api_ctrl*)

Suspend Reception

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Functionality not supported by this driver instance |

### ◆ R_SAU_UART_ReceiveResume()

fsp_err_t R_SAU_UART_ReceiveResume ( uart_ctrl_t *const *p_api_ctrl*)

Resume Reception

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Functionality not supported by this driver instance |

### 5.2.6.29 UART (r_sci_b_uart)

Modules » Connectivity

#### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_B_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_B_UART_Close (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SCI_B_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SCI_B_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_B_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_SCI_B_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info) |
| fsp_err_t | R_SCI_B_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort) |
| fsp_err_t | R_SCI_B_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes) |
| fsp_err_t | R_SCI_B_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, sci_b_baud_setting_t *const p_baud_setting) |
| fsp_err_t | R_SCI_B_UART_ReceiveSuspend (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_B_UART_ReceiveResume (uart_ctrl_t *const p_api_ctrl) |

#### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.

# Overview

#### Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- CTS/RTS hardware flow control (with an associated pin)
- RS-485 Half Duplex driver support with external RS-485 transceiver
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

# Configuration

### Build Time Configurations for r_sci_b_uart

The following build time configurations are defined in fsp_cfg/r_sci_b_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| FIFO Support | • Enable<br>• Disable | Disable | Enable FIFO support for the SCI_UART module. |
| DTC Support | • Enable<br>• Disable | Disable | Enable DTC support for the SCI_UART module. |
| Flow Control Support | • Enable<br>• Disable | Disable | Enable RS232 and RS-485 flow control support using a user provided pin. |

### Configurations for Connectivity > UART (r_sci_b_uart)

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sci_b_uart). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_uart0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SCI channel. |
| Data Bits | • 8bits<br>• 7bits<br>• 9bits | 8bits | Select the number of bits per word. |

| | | | |
|---|---|---|---|
| Parity | • None<br>• Odd<br>• Even | None | Select the parity mode. |
| Stop Bits | • 1bit<br>• 2bits | 1bit | Select the number of stop bits. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 115200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated sci_b_baud_setting_t structure. |
| Baud Rate Modulation | • Disabled<br>• Enabled | Disabled | Enabling baud rate modulation reduces the percent error of the actual baud rate with respect to the requested baud rate. It does this by modulating the number of cycles per clock, so some bits are slightly longer than others. |
| Max Error (%) | Must be a valid non-negative integer with a maximum configurable value of 100 | 5 | Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings.<br><br>NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated sci_b_baud_setting_t |

| | | | |
|---|---|---|---|
| | | | structure. |
| Flow Control | | | |
| CTS/RTS Selection | MCU Specific Options | | Select either CTS or RTS function on the CTSn_RTSn pin of SCI channel n or select CTS function on CTSn pin and RTS function on CTSn_RTSn pin of SCI channel n (Available on selected MCUs and channels). |
| Software RTS Port | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin port for the MCU. |
| Software RTS Pin | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin for the MCU. |
| Extra | | | |
| Extra > RS-485 | | | |
| DE Pin | • Disable<br>• Enable | Disable | Enable or disable the DE pin for use in RS-485 half-duplex mode. |
| DE Pin Polarity | • Active Low<br>• Active High | Active High | Select the polarity of the DE pin. |
| DE Pin Assertion Time | Must be a valid integer greater than 0 and less than or equal to 31. | 1 | Configure the time between assertion of the DE pin and the start of a write transfer. The time is specified in multiples of the SCI base clock period. |
| DE Pin Negation Time | Must be a valid integer greater than 0 and less than or equal to 31. | 1 | Configure the time between the end of a write transfer and the negation of the DE pin. The time is specified in multiples of the SCI base clock period. |
| Clock Source | • Internal Clock<br>• Internal Clock With Output on SCK<br>• External Clock 8x baud rate<br>• External Clock | Internal Clock | Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin. |

| | | | |
|---|---|---|---|
| | 16x baud rate | | |
| Start bit detection | • Falling Edge<br>• Low Level | Falling Edge | Start bit detected as falling edge or low level. |
| Noise Filter | • Enable<br>• Disable | Disable | Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits. |
| Receive FIFO Trigger Level | Refer to the RA Configuration tool for available options. | Max | Unused if the channel has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when FIFO is full or after 15 bit times with no data (fewer interrupts). |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Data Empty Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA6T2 | SCISPICLK |
| RA8D1 | SCICLK |
| RA8D2 | SCICLK |

| | |
|---|---|
| RA8E1 | SCICLK |
| RA8E2 | SCICLK |
| RA8M1 | SCICLK |
| RA8M2 | SCICLK |
| RA8P1 | SCICLK |
| RA8T1 | SCICLK |
| RA8T2 | SCICLK |

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

### Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS can be controlled by the hardware. CTS or RTS or both (CTS and RTS) can be controlled by the hardware When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

## Usage Notes

- When configured for Hardware CTS and Software RTS the configured flow control pin will be used for RTS. The pin will be set high inside of the receive ISR while data is being read. It will be set low when all data is read.
- When configured for Hardware CTS and Hardware RTS the CSTn_RTSn pin will be used for RTS function and the CTSn pin will be used for CTS function on channel n.
- The driver will follow correct hardware flow control function when CTSn_RTSn pin is connected to CTSn pin when "Hardware CTS and Hardware RTS" flow control is selected. The data will still be transferred when CTSn_RTSn and CTSn are disconnected as the CTSn pin is internally pulled low on the hardware when CTSn pin is configured as a peripheral pin for SCI module. Do not configure CTSn pin if the hardware flow control is not desired.

### Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.

### DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_SCI_B_UART_Read(). The logic required to combine bytes received through R_SCI_B_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

# Examples

### SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_b_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_B_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
```

```
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                  g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                  uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                  *p_dest             = (uint16_t) p_args->data;
                  g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
```

```
 default:
      {
      }
    }
}
```

### SCI UART Baud Set Example

```
#define SCI_B_UART_BAUDRATE_19200 (19200)
#define SCI_B_UART_BAUDRATE_ERROR_PERCENT_5 (5000)
void r_sci_b_uart_baud_example (void)
{
 sci_b_baud_setting_t baud_setting;
    uint32_t             baud_rate                = SCI_B_UART_BAUDRATE_19200;
 bool                enable_bitrate_modulation = false;
    uint32_t             error_rate_x_1000         =
SCI_B_UART_BAUDRATE_ERROR_PERCENT_5;
 fsp_err_t err = R_SCI_B_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | sci_b_uart_instance_ctrl_t |
| struct | sci_b_baud_setting_t |
| struct | sci_b_uart_rs485_setting_t |
| struct | sci_b_uart_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | sci_b_clk_src_t |
| enum | sci_b_uart_flow_control_t |
| enum | sci_b_uart_rx_fifo_trigger_t |

| | |
|---|---|
| enum | sci_b_uart_start_bit_detect_t |
| enum | sci_b_uart_noise_cancellation_t |
| enum | sci_b_uart_rs485_enable_t |
| enum | sci_b_uart_rs485_de_polarity_t |

## Data Structure Documentation

### ◆ sci_b_uart_instance_ctrl_t

| struct sci_b_uart_instance_ctrl_t |
|---|
| UART instance control block. |

### ◆ sci_b_baud_setting_t

| struct sci_b_baud_setting_t |
|---|
| Register settings to achieve a desired baud rate and modulation duty. |

### ◆ sci_b_uart_rs485_setting_t

| struct sci_b_uart_rs485_setting_t | | |
|---|---|---|
| Configuration settings for controlling the DE signal for RS-485. | | |
| Data Fields | | |
| sci_b_uart_rs485_enable_t | enable | Enable the DE signal. |
| sci_b_uart_rs485_de_polarity_t | polarity | DE signal polarity. |
| uint8_t | assertion_time: 5 | Time in baseclock units after assertion of the DE signal and before the start of the write transfer. |
| uint8_t | negation_time: 5 | Time in baseclock units after the end of a write transfer and before the DE signal is negated. |

### ◆ sci_b_uart_extended_cfg_t

| struct sci_b_uart_extended_cfg_t | | |
|---|---|---|
| UART on SCI device Configuration | | |
| Data Fields | | |
| sci_b_clk_src_t | clock | The source clock for the baud-rate generator. If internal optionally output baud rate on SCK. |

| | | |
|---|---|---|
| sci_b_uart_start_bit_detect_t | rx_edge_start | Start reception on falling edge. |
| sci_b_uart_noise_cancellation_t | noise_cancel | Noise cancellation setting. |
| sci_b_baud_setting_t * | p_baud_setting | Register settings for a desired baud rate. |
| sci_b_uart_rx_fifo_trigger_t | rx_fifo_trigger | Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used. |
| bsp_io_port_pin_t | flow_control_pin | UART Driver Enable pin. |
| sci_b_uart_flow_control_t | flow_control | CTS/RTS function of the SSn pin. |
| sci_b_uart_rs485_setting_t | rs485_setting | RS-485 settings. |

## Enumeration Type Documentation

### ◆ sci_b_clk_src_t

| enum sci_b_clk_src_t | |
|---|---|
| Enumeration for SCI clock source | |
| Enumerator | |
| SCI_B_UART_CLOCK_INT | Use internal clock for baud generation. |
| SCI_B_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT | Use internal clock for baud generation and output on SCK. |
| SCI_B_UART_CLOCK_EXT8X | Use external clock 8x baud rate. |
| SCI_B_UART_CLOCK_EXT16X | Use external clock 16x baud rate. |

### ◆ sci_b_uart_flow_control_t

| enum sci_b_uart_flow_control_t | |
|---|---|
| UART flow control mode definition | |
| Enumerator | |
| SCI_B_UART_FLOW_CONTROL_RTS | Use CTSn_RTSn pin for RTS. |
| SCI_B_UART_FLOW_CONTROL_CTS | Use CTSn_RTSn pin for CTS. |
| SCI_B_UART_FLOW_CONTROL_HARDWARE_CTSRTS | Use CTSn pin for CTS, CTSn_RTSn pin for RTS. |
| SCI_B_UART_FLOW_CONTROL_CTSRTS | Use SCI pin for CTS, external pin for RTS. |

### ◆ sci_b_uart_rx_fifo_trigger_t

| enum sci_b_uart_rx_fifo_trigger_t | |
|---|---|
| Receive FIFO trigger configuration. | |
| Enumerator | |
| SCI_B_UART_RX_FIFO_TRIGGER_1 | Callback after each byte is received without buffering. |
| SCI_B_UART_RX_FIFO_TRIGGER_2 | Callback when FIFO having 2 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_3 | Callback when FIFO having 3 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_4 | Callback when FIFO having 4 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_5 | Callback when FIFO having 5 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_6 | Callback when FIFO having 6 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_7 | Callback when FIFO having 7 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_8 | Callback when FIFO having 8 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_9 | Callback when FIFO having 9 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_10 | Callback when FIFO having 10 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_11 | Callback when FIFO having 11 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_12 | Callback when FIFO having 12 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_13 | Callback when FIFO having 13 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_14 | Callback when FIFO having 14 bytes. |
| SCI_B_UART_RX_FIFO_TRIGGER_MAX | Callback when FIFO is full or after 15 bit times with no data (fewer interrupts) |

### ◆ sci_b_uart_start_bit_detect_t

| enum sci_b_uart_start_bit_detect_t | |
|---|---|
| Asynchronous Start Bit Edge Detection configuration. | |
| Enumerator | |
| SCI_B_UART_START_BIT_LOW_LEVEL | Detect low level on RXDn pin as start bit. |
| SCI_B_UART_START_BIT_FALLING_EDGE | Detect falling level on RXDn pin as start bit. |

### ◆ sci_b_uart_noise_cancellation_t

| enum sci_b_uart_noise_cancellation_t | |
|---|---|
| Noise cancellation configuration. | |
| Enumerator | |
| SCI_B_UART_NOISE_CANCELLATION_DISABLE | Disable noise cancellation. |
| SCI_B_UART_NOISE_CANCELLATION_ENABLE | Enable noise cancellation. |

### ◆ sci_b_uart_rs485_enable_t

| enum sci_b_uart_rs485_enable_t | |
|---|---|
| RS-485 Enable/Disable. | |
| Enumerator | |
| SCI_B_UART_RS485_DISABLE | RS-485 disabled. |
| SCI_B_UART_RS485_ENABLE | RS-485 enabled. |

### ◆ sci_b_uart_rs485_de_polarity_t

| enum sci_b_uart_rs485_de_polarity_t | |
|---|---|
| The polarity of the RS-485 DE signal. | |
| Enumerator | |
| SCI_B_UART_RS485_DE_POLARITY_HIGH | The DE signal is high when a write transfer is in progress. |
| SCI_B_UART_RS485_DE_POLARITY_LOW | The DE signal is low when a write transfer is in progress. |

## Function Documentation

### ◆ R_SCI_B_UART_Open()

fsp_err_t R_SCI_B_UART_Open ( uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg* )

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Flow control is enabled but flow control pin is not defined. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SCI_B_UART_Close()

fsp_err_t R_SCI_B_UART_Close ( uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_B_UART_Read()

fsp_err_t R_SCI_B_UART_Read ( uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Destination address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |
| FSP_ERR_UNSUPPORTED | SCI_B_UART_CFG_RX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_B_UART_Open call, p_dest must be aligned 16-bit boundary.*

### ◆ R_SCI_B_UART_Write()

fsp_err_t R_SCI_B_UART_Write ( uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission finished successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Source address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A UART transmission is in progress |
| FSP_ERR_UNSUPPORTED | SCI_B_UART_CFG_TX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_B_UART_Open call, p_src must be aligned on a 16-bit boundary.*

### ◆ R_SCI_B_UART_CallbackSet()

fsp_err_t R_SCI_B_UART_CallbackSet ( uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void *const *p_context*, uart_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_B_UART_BaudSet()

fsp_err_t R_SCI_B_UART_BaudSet ( uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a sci_b_baud_setting_t structure. Implements uart_api_t::baudSet

**Warning**

This terminates any in-progress transmission.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL or the UART is not configured to use the internal clock. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_B_UART_InfoGet()

fsp_err_t R_SCI_B_UART_InfoGet ( uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info* )

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_B_UART_Abort()

fsp_err_t R_SCI_B_UART_Abort ( uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort* )

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SCI_B_UART_ReadStop()

fsp_err_t R_SCI_B_UART_ReadStop ( uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes* )

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SCI_B_UART_BaudCalculate()

fsp_err_t R_SCI_B_UART_BaudCalculate ( uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, sci_b_baud_setting_t *const *p_baud_setting* )

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

**Parameters**

| | | |
|---|---|---|
| [in] | baudrate | Baud rate [bps]. For example, 19200, 57600, 115200, etc. |
| [in] | bitrate_modulation | Enable bitrate modulation |
| [in] | baud_rate_error_x_1000 | Max baud rate error. At most <baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%). |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully |
| FSP_ERR_ASSERTION | Null pointer |
| FSP_ERR_INVALID_ARGUMENT | Baud rate is '0', error in calculated baud rate is larger than requested max error, or requested max error in baud rate is larger than 15%. |

### ◆ R_SCI_B_UART_ReceiveSuspend()

fsp_err_t R_SCI_B_UART_ReceiveSuspend ( uart_ctrl_t *const *p_api_ctrl*)

Suspend Reception

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Functionality not supported by this driver instance |

### ◆ R_SCI_B_UART_ReceiveResume()

| fsp_err_t R_SCI_B_UART_ReceiveResume ( uart_ctrl_t *const *p_api_ctrl*) |
|---|
| Resume Reception<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Functionality not supported by this driver instance |

## 5.2.6.30 UART (r_sci_uart)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_SCI_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCI_UART_Close (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_SCI_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_SCI_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SCI_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_SCI_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info) |
| fsp_err_t | R_SCI_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort) |
| fsp_err_t | R_SCI_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes) |
| fsp_err_t | R_SCI_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, baud_setting_t *const p_baud_setting) |

| | |
|---|---|
| fsp_err_t | R_SCI_UART_ReceiveSuspend (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_SCI_UART_ReceiveResume (uart_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.

# Overview

### Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- CTS/RTS hardware flow control (with an associated pin)
- RS-485 Half Duplex driver support with external RS-485 transceiver
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels
- IrDA is supported on devices that have IrDA interface

# Configuration

### Build Time Configurations for r_sci_uart

The following build time configurations are defined in fsp_cfg/r_sci_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| FIFO Support | • Enable<br>• Disable | Disable | Enable FIFO support for the SCI_UART module. |
| DTC Support | • Enable<br>• Disable | Disable | Enable DTC support for the SCI_UART module. |
| Flow Control Support | • Enable<br>• Disable | Disable | Enable RS232 and RS-485 flow control support using a user provided pin. |
| RS-485 Support | • Enable<br>• Disable | Disable | Enable support for controlling the RS-485 DE pin. |

| | | | |
|---|---|---|---|
| IrDA Support | • Enabled<br>• Disabled | Disabled | Enable support for IrDA pulse encoding/decoding. |

**Configurations for Connectivity > UART (r_sci_uart)**

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sci_uart). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_uart0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the SCI channel. |
| Data Bits | • 8bits<br>• 7bits<br>• 9bits | 8bits | Select the number of bits per word. |
| Parity | • None<br>• Odd<br>• Even | None | Select the parity mode. |
| Stop Bits | • 1bit<br>• 2bits | 1bit | Select the number of stop bits. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 115200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated baud_setting_t structure.<br><br>For RA4 and RA6 MCUs, refer the datasheet to calculate baud rate considering the SEMR.ABCSE bit restriction. |
| Baud Rate Modulation | • Disabled<br>• Enabled | Disabled | Enabling baud rate modulation reduces the |

| | | | |
|---|---|---|---|
| | | | percent error of the actual baud rate with respect to the requested baud rate. It does this by modulating the number of cycles per clock, so some bits are slightly longer than others. |
| Max Error (%) | Must be a valid non-negative integer with a maximum configurable value of 100 | 5 | Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings.<br><br>NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated baud_setting_t structure. |
| Flow Control | | | |
| CTS/RTS Selection | MCU Specific Options | | Select either CTS or RTS function on the CTSn_RTSn pin of SCI channel n or select CTS function on CTSn pin and RTS function on CTSn_RTSn pin of SCI channel n (Available on selected MCUs and channels). |
| Software RTS Port | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin port for the MCU. |
| Software RTS Pin | Refer to the RA Configuration tool for available options. | Disabled | Specify the flow control pin for the MCU. |
| Extra | | | |
| Extra > IrDA | | | |
| Enable | • Disabled | Disabled | Enable IrDA on this |

| | | | |
|---|---|---|---|
| | • Enabled | | channel. |
| RXD Polarity Switching | • Normal<br>• Inverted | Normal | IRRXD Pulse polarity (Normal: High Pulse=0) and (Inverted: Low Pulse=0).. |
| TXD Polarity Switching | • Normal<br>• Inverted | Normal | IRTXD Pulse polarity (Normal: High Pulse 0) and (Inverted: Low Pulse=0). |
| Extra > RS-485 | | | |
| DE Pin | • Disable<br>• Enable | Disable | Enable or disable the DE pin for use in RS-485 half-duplex mode. |
| DE Pin Polarity | • Active Low<br>• Active High | Active High | Select the polarity of the DE pin. |
| DE Port Number | Refer to the RA Configuration tool for available options. | Disabled | GPIO output port number to use for generating the DE signal. |
| DE Pin Number | Refer to the RA Configuration tool for available options. | Disabled | GPIO output pin number to use for generating the DE signal. |
| Clock Source | • Internal Clock<br>• Internal Clock With Output on SCK<br>• External Clock 8x baud rate<br>• External Clock 16x baud rate | Internal Clock | Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin. |
| Start bit detection | • Falling Edge<br>• Low Level | Falling Edge | Start bit detected as falling edge or low level. |
| Noise Filter | • Enable<br>• Disable | Disable | Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits. |
| Receive FIFO Trigger Level | Refer to the RA Configuration tool for available options. | Max | Unused if the channel has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when |

| | | | |
|---|---|---|---|
| | | | FIFO is full or after 15 bit times with no data (fewer interrupts). |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Data Empty Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Transmit End Interrupt Priority | MCU Specific Options | | Select the transmit end interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA2A1 | PCLKB |
| RA2A2 | PCLKB |
| RA2E1 | PCLKB |
| RA2E2 | PCLKB |
| RA2E3 | PCLKB |
| RA2L1 | PCLKB |
| RA2L2 | PCLKB |
| RA2T1 | PCLKB |
| RA4C1 | PCLKA |
| RA4E1 | PCLKA |
| RA4E2 | PCLKA |
| RA4L1 | PCLKA |
| RA4M1 | PCLKA |
| RA4M2 | PCLKA |

| | |
|---|---|
| RA4M3 | PCLKA |
| RA4T1 | PCLKA |
| RA4W1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M1 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T1 | PCLKA |
| RA6T3 | PCLKA |

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

### Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS or both (CTS and RTS) can be controlled by the hardware. Some MCUs support hardware flow control for both CTS and RTS on some channels. Some MCUs and channels support hardware flow control for either CTS or RTS but not both. If both are desired a GPIO pin can be used for RTS. When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

# Usage Notes

- When configured for Hardware CTS and Software RTS the configured flow control pin will be used for RTS. The pin will be set high inside of the receive ISR while data is being read. It will be set low when all data is read.
- When configured for Hardware CTS and Hardware RTS the CSTn_RTSn pin will be used for RTS function and the CTSn pin will be used for CTS function on channel n.
- The driver will follow correct hardware flow control function when CTSn_RTSn pin is connected to CTSn pin when "Hardware CTS and Hardware RTS" flow control is selected. The data will still be transferred when CTSn_RTSn and CTSn are disconnected as the CTSn pin is internally pulled low on the hardware when CTSn pin is configured as a peripheral pin for SCI module. Do not configure CTSn pin if the hardware flow control is not desired.

### Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.
- The R_SCI_UART_BaudCalculate() and R_SCI_UART_Open() APIs are used to obtain the baud rate register settings for a specific baudrate. On RA4 and RA6 MCU groups, there is a restriction on SEMR.ABSCE bit for certain channels (refer to 'SEMR : Serial Extended Mode' Register section of MCU User Manual). The API does not enforce this restriction and it may

return invalid settings.

### DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_SCI_UART_Read(). The logic required to combine bytes received through R_SCI_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

### IrDA Limitations

- The IrDA interface works in cooperation with a particular SCI channel based on the IrDA standard 1.0.
- If optics placement yields the echo of transmitted data on the receiver, the application should disable the RX irq during transmission.
- R_SCI_UART_BaudCalculate shall not be called for IrDA baud rate calculation. The correct use is to create the baud rate by using the configurator. If run-time changes to baud rate are required, multiple instances of different baud rates should be created with the configurator.

## Examples

### SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
```

```
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
     {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
     {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
     {
                 g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
     }
 else
     {
                 uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                 *p_dest             = (uint16_t) p_args->data;
```

```
                g_out_of_band_index += 2;
     }
     }
 break;
     }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
     {
           g_receive_complete = 1;
 break;
     }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
     {
           g_transfer_complete = 1;
 break;
     }
 default:
     {
     }
   }
}
```

### SCI UART Baud Set Example

```
#define SCI_UART_BAUDRATE_19200 (19200)
#define SCI_UART_BAUDRATE_ERROR_PERCENT_5 (5000)
void r_sci_uart_baud_example (void)
{
 baud_setting_t baud_setting;
    uint32_t       baud_rate                = SCI_UART_BAUDRATE_19200;
 bool          enable_bitrate_modulation = false;
    uint32_t       error_rate_x_1000         = SCI_UART_BAUDRATE_ERROR_PERCENT_5;
 fsp_err_t err = R_SCI_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
```

```
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | sci_uart_instance_ctrl_t |
| struct | baud_setting_t |
| struct | sci_uart_rs485_setting_t |
| struct | sci_uart_irda_setting_t |
| struct | sci_uart_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | sci_clk_src_t |
| enum | sci_uart_flow_control_t |
| enum | sci_uart_rx_fifo_trigger_t |
| enum | sci_uart_start_bit_t |
| enum | sci_uart_noise_cancellation_t |
| enum | sci_uart_rs485_enable_t |
| enum | sci_uart_rs485_de_polarity_t |
| enum | sci_uart_irda_enable_t |
| enum | sci_uart_irda_polarity_t |

## Data Structure Documentation

### ◆ sci_uart_instance_ctrl_t

| |
|---|
| struct sci_uart_instance_ctrl_t |
| UART instance control block. |

### ◆ baud_setting_t

| |
|---|
| struct baud_setting_t |
| |

| Register settings to achieve a desired baud rate and modulation duty. | | |
|---|---|---|
| Data Fields | | |
| union baud_setting_t | __unnamed__ | |
| uint8_t | cks: 2 | CKS value to get divisor (CKS = N) |
| uint8_t | brr | Bit Rate Register setting. |
| uint8_t | mddr | Modulation Duty Register setting. |

### ◆ sci_uart_rs485_setting_t

| struct sci_uart_rs485_setting_t | | |
|---|---|---|
| Configuration settings for controlling the DE signal for RS-485. | | |
| Data Fields | | |
| sci_uart_rs485_enable_t | enable | Enable the DE signal. |
| sci_uart_rs485_de_polarity_t | polarity | DE signal polarity. |
| bsp_io_port_pin_t | de_control_pin | UART Driver Enable pin. |

### ◆ sci_uart_irda_setting_t

| struct sci_uart_irda_setting_t |
|---|
| Configuration settings for IrDA interface. |

### ◆ sci_uart_extended_cfg_t

| struct sci_uart_extended_cfg_t | | |
|---|---|---|
| UART on SCI device Configuration | | |
| Data Fields | | |
| sci_clk_src_t | clock | The source clock for the baud-rate generator. If internal optionally output baud rate on SCK. |
| sci_uart_start_bit_t | rx_edge_start | Start reception on falling edge. |
| sci_uart_noise_cancellation_t | noise_cancel | Noise cancellation setting. |
| baud_setting_t * | p_baud_setting | Register settings for a desired baud rate. |
| sci_uart_rx_fifo_trigger_t | rx_fifo_trigger | Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used. |
| bsp_io_port_pin_t | flow_control_pin | UART Driver Enable pin. |
| sci_uart_flow_control_t | flow_control | CTS/RTS function of the SSn pin. |

| | | |
|---|---|---|
| sci_uart_rs485_setting_t | rs485_setting | RS-485 settings. |
| sci_uart_irda_setting_t | irda_setting | IrDA settings. |

## Enumeration Type Documentation

### ◆ sci_clk_src_t

| enum sci_clk_src_t | |
|---|---|
| Enumeration for SCI clock source | |
| Enumerator | |
| SCI_UART_CLOCK_INT | Use internal clock for baud generation. |
| SCI_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT | Use internal clock for baud generation and output on SCK. |
| SCI_UART_CLOCK_EXT8X | Use external clock 8x baud rate. |
| SCI_UART_CLOCK_EXT16X | Use external clock 16x baud rate. |

### ◆ sci_uart_flow_control_t

| enum sci_uart_flow_control_t | |
|---|---|
| UART flow control mode definition | |
| Enumerator | |
| SCI_UART_FLOW_CONTROL_RTS | Use SCI pin for RTS. |
| SCI_UART_FLOW_CONTROL_CTS | Use SCI pin for CTS. |
| SCI_UART_FLOW_CONTROL_CTSRTS | Use SCI pin for CTS, external pin for RTS. |
| SCI_UART_FLOW_CONTROL_HARDWARE_CTSRTS | Use CTSn_RTSn pin for RTS and CTSn pin for CTS. Available only for some channels on selected MCUs. See hardware manual for channel specific options. |

### ◆ sci_uart_rx_fifo_trigger_t

| enum sci_uart_rx_fifo_trigger_t | |
|---|---|
| Receive FIFO trigger configuration. | |
| Enumerator | |
| SCI_UART_RX_FIFO_TRIGGER_1 | Callback after each byte is received without buffering. |
| SCI_UART_RX_FIFO_TRIGGER_2 | Callback when FIFO having 2 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_3 | Callback when FIFO having 3 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_4 | Callback when FIFO having 4 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_5 | Callback when FIFO having 5 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_6 | Callback when FIFO having 6 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_7 | Callback when FIFO having 7 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_8 | Callback when FIFO having 8 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_9 | Callback when FIFO having 9 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_10 | Callback when FIFO having 10 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_11 | Callback when FIFO having 11 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_12 | Callback when FIFO having 12 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_13 | Callback when FIFO having 13 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_14 | Callback when FIFO having 14 bytes. |
| SCI_UART_RX_FIFO_TRIGGER_MAX | Callback when FIFO is full or after 15 bit times with no data (fewer interrupts) |

### ◆ sci_uart_start_bit_t

| enum sci_uart_start_bit_t | |
|---|---|
| Asynchronous Start Bit Edge Detection configuration. | |
| Enumerator | |
| SCI_UART_START_BIT_LOW_LEVEL | Detect low level on RXDn pin as start bit. |
| SCI_UART_START_BIT_FALLING_EDGE | Detect falling level on RXDn pin as start bit. |

### ◆ sci_uart_noise_cancellation_t

| enum sci_uart_noise_cancellation_t | |
|---|---|
| Noise cancellation configuration. | |
| Enumerator | |
| SCI_UART_NOISE_CANCELLATION_DISABLE | Disable noise cancellation. |
| SCI_UART_NOISE_CANCELLATION_ENABLE | Enable noise cancellation. |

### ◆ sci_uart_rs485_enable_t

| enum sci_uart_rs485_enable_t | |
|---|---|
| RS-485 Enable/Disable. | |
| Enumerator | |
| SCI_UART_RS485_DISABLE | RS-485 disabled. |
| SCI_UART_RS485_ENABLE | RS-485 enabled. |

### ◆ sci_uart_rs485_de_polarity_t

| enum sci_uart_rs485_de_polarity_t | |
|---|---|
| The polarity of the RS-485 DE signal. | |
| Enumerator | |
| SCI_UART_RS485_DE_POLARITY_HIGH | The DE signal is high when a write transfer is in progress. |
| SCI_UART_RS485_DE_POLARITY_LOW | The DE signal is low when a write transfer is in progress. |

### ◆ sci_uart_irda_enable_t

| enum sci_uart_irda_enable_t | |
|---|---|
| IrDA Enable/Disable. | |
| Enumerator | |
| SCI_UART_IRDA_DISABLED | IrDA disabled. |
| SCI_UART_IRDA_ENABLED | IrDA enabled. |

### ◆ sci_uart_irda_polarity_t

| enum sci_uart_irda_polarity_t | |
|---|---|
| IrDA Polarity Switching. | |
| Enumerator | |
| SCI_UART_IRDA_POLARITY_NORMAL | IrDA Tx/Rx polarity not inverted. |
| SCI_UART_IRDA_POLARITY_INVERTED | IrDA Tx/Rx polarity inverted. |

## Function Documentation

### ◆ R_SCI_UART_Open()

fsp_err_t R_SCI_UART_Open ( uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg* )

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Flow control is enabled but flow control pin is not defined or selected channel does not support "Hardware CTS and Hardware RTS" flow control. (or) restricted channel is selected. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_CHANNEL | IrDA is requested for a channel that does not support IrDA. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_SCI_UART_Close()

fsp_err_t R_SCI_UART_Close ( uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_UART_Read()

fsp_err_t R_SCI_UART_Read ( uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Destination address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |
| FSP_ERR_UNSUPPORTED | SCI_UART_CFG_RX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_UART_Open call, p_dest must be aligned 16-bit boundary.*

### ◆ R_SCI_UART_Write()

fsp_err_t R_SCI_UART_Write ( uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission finished successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_INVALID_ARGUMENT | Source address or data size is not valid for 9-bit mode. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A UART transmission is in progress |
| FSP_ERR_UNSUPPORTED | SCI_UART_CFG_TX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

***Note***

*If 9-bit data length is specified at R_SCI_UART_Open call, p_src must be aligned on a 16-bit boundary.*

### ◆ R_SCI_UART_CallbackSet()

fsp_err_t R_SCI_UART_CallbackSet ( uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void *const *p_context*, uart_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SCI_UART_BaudSet()

fsp_err_t R_SCI_UART_BaudSet ( uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a baud_setting_t structure. Implements uart_api_t::baudSet

**Warning**

This terminates any in-progress transmission.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL or the UART is not configured to use the internal clock. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Restricted channel is selected. |

### ◆ R_SCI_UART_InfoGet()

fsp_err_t R_SCI_UART_InfoGet ( uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info* )

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_SCI_UART_Abort()

fsp_err_t R_SCI_UART_Abort ( uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort* )

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SCI_UART_ReadStop()

fsp_err_t R_SCI_UART_ReadStop ( uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes* )

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

### ◆ R_SCI_UART_BaudCalculate()

fsp_err_t R_SCI_UART_BaudCalculate ( uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, baud_setting_t *const *p_baud_setting* )

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

***Note***

*For limitations of this API, refer to the 'Limitations' section of r_sci_uart module in FSP User Manual.*

**Parameters**

| | | |
|---|---|---|
| [in] | baudrate | Baud rate [bps]. For example, 19200, 57600, 115200, etc. |
| [in] | bitrate_modulation | Enable bitrate modulation |
| [in] | baud_rate_error_x_1000 | Max baud rate error. At most <baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%). |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully |
| FSP_ERR_ASSERTION | Null pointer |
| FSP_ERR_INVALID_ARGUMENT | Baud rate is '0', error in calculated baud rate is larger than requested max error, or requested max error in baud rate is larger than 15%. |

### ◆ R_SCI_UART_ReceiveSuspend()

fsp_err_t R_SCI_UART_ReceiveSuspend ( uart_ctrl_t *const *p_api_ctrl*)

Suspend Reception

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Functionality not supported by this driver instance |

### ◆ R_SCI_UART_ReceiveResume()

| fsp_err_t R_SCI_UART_ReceiveResume ( uart_ctrl_t *const *p_api_ctrl*) |
|---|
| Resume Reception<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Functionality not supported by this driver instance |

## 5.2.6.31 UART (r_uarta)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_UARTA_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) |
| fsp_err_t | R_UARTA_Close (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_UARTA_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_UARTA_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| fsp_err_t | R_UARTA_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_UARTA_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting) |
| fsp_err_t | R_UARTA_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info) |
| fsp_err_t | R_UARTA_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort) |
| fsp_err_t | R_UARTA_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *p_remaining_bytes) |
| fsp_err_t | R_UARTA_BaudCalculate (uint32_t baudrate, uint32_t baud_rate_percent_error_x1000, uarta_clock_source_t clock_source, uarta_baud_setting_t *const p_baud_setting) |

| | |
|---|---|
| fsp_err_t | R_UARTA_ReceiveSuspend (uart_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_UARTA_ReceiveResume (uart_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the UARTA peripheral on RA MCUs. This module implements the UART Interface.

# Overview

### Features

The UARTA module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Baud-rate change at run-time
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- MSB or LSB first transfer selectable
- Inversion control of communication logic level provided
- Clock output function (Not available on all MCUs)

# Configuration

### Build Time Configurations for r_uarta

The following build time configurations are defined in fsp_cfg/r_uarta_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DTC Support | MCU Specific Options | | Enable DTC support for the UARTA module. |
| Receive Error Interrupt Mode | • Disabled<br>• Enabled | Enabled | Selection receive interrupt mode . Disabled: The UARTA0_ERRI interrupt is generated when a reception error occurs. Enabled: The UARTA0_RXI interrupt is generated when a reception error occurs. |

### Configurations for Connectivity > UART (r_uarta)

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_uarta). Non-secure callable guard functions can be generated for this module by right clicking the module in the

RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_uart0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Select the UARTA channel. |
| Data Bits | • 8bits<br>• 7bits<br>• 5bits | 8bits | Select the number of bits per word. |
| Parity | • None<br>• Zero<br>• Odd<br>• Even | None | Select the parity mode. |
| Stop Bits | • 1bit<br>• 2bits | 1bit | Select the number of stop bits.<br>Note: For the receive data, only the first 1 bit of the stop bits is checked regardless of the stop bit length. |
| Baud | | | |
| Baud Rate | Value must be an integer greater than 0 | 115200 | Enter the desired baud rate.<br><br>If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated baud_setting_t structure. |
| Extra | | | |
| Transfer Order | • LSB first<br>• MSB first | LSB first | Selection of the transmission/reception order. |
| Transfer level | • Positive logic<br>• Negative logic | Positive logic | Selection of the transmission/reception level. |
| Clock output | MCU Specific Options | | Enable Clock output |

| | | | |
|---|---|---|---|
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Receive Interrupt Priority | MCU Specific Options | | Select the receive interrupt priority. |
| Transmit Interrupt Priority | MCU Specific Options | | Select the transmit interrupt priority. |
| Error Interrupt Priority | MCU Specific Options | | Select the error interrupt priority. |

### Clock Configuration

The baud-rate clock generator can be selected from the clock source MOSC, HOCO, MOCO, SOSC/LOCO.

### Pin Configuration

This module uses TXDA and RXDA pin to communicate to external devices.

# Usage Notes

### Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.
- The UARTA module does not have CTS or RTS signals.
- UARTA 0ch and 1ch cannot be configured for secure and non-secure usage simultaneously.

### DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_UARTA_Read(). The logic required to combine bytes received through R_UARTA_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.

# Examples

### UARTA Example

```
uint8_t          g_dest[TRANSFER_LENGTH];
```

```
uint8_t          g_src[TRANSFER_LENGTH];
uint8_t          g_out_of_band_received[TRANSFER_LENGTH];
volatile uint32_t g_transfer_complete = 0;
volatile uint32_t g_receive_complete = 0;
uint32_t          g_out_of_band_index = 0;
void r_uarta_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_UARTA_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_UARTA_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_UARTA_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
```

```
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                  g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                  uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                  *p_dest             = (uint16_t) p_args->data;
                  g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
```

```
    }
}
```

### UARTA Baud Set Example

```
#define UARTA_BAUDRATE_19200 (19200)
#define UARTA_BAUDRATE_ERROR_PERCENT_4740 (4740)
void r_uarta_baud_example (void)
{
 uarta_baud_setting_t baud_setting;
    uint32_t             baud_rate         = UARTA_BAUDRATE_19200;
    uint32_t             error_rate_x_1000 = UARTA_BAUDRATE_ERROR_PERCENT_4740;
 uarta_clock_source_t clock_source      = UARTA_CLOCK_SOURCE_HOCO;
 fsp_err_t err = R_UARTA_BaudCalculate(baud_rate, error_rate_x_1000, clock_source,
&baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_UARTA_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | uarta_baud_setting_t |
| struct | uarta_extended_cfg_t |
| struct | uarta_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | uarta_clock_source_t |
| enum | uarta_clock_div_t |
| enum | uarta_clock_out_t |
| enum | uarta_dir_bit_t |
| enum | uarta_alv_bit_t |

## Data Structure Documentation

### ◆ uarta_baud_setting_t

| struct uarta_baud_setting_t | | |
|---|---|---|
| Register settings to achieve a desired baud rate and modulation duty. | | |
| Data Fields | | |
| union uarta_baud_setting_t | __unnamed__ | |
| uint8_t | brgca | Baud rate generator control setting. |
| uint16_t | delay_time | Delay time (us) required to enable TX at open. |

### ◆ uarta_extended_cfg_t

| struct uarta_extended_cfg_t | | |
|---|---|---|
| UART on UARTA device Configuration | | |
| Data Fields | | |
| uarta_dir_bit_t | transfer_dir | Transmission/reception order configuration. |
| uarta_alv_bit_t | transfer_level | Transmission/reception level configuration. |
| uarta_clock_out_t | clock_output | Disable/Enable clock output. |
| uarta_baud_setting_t * | p_baud_setting | Register settings for a desired baud rate. |

### ◆ uarta_instance_ctrl_t

| struct uarta_instance_ctrl_t |
|---|
| UARTA instance control block. |

## Enumeration Type Documentation

### ◆ uarta_clock_source_t

| enum uarta_clock_source_t | |
|---|---|
| Enumeration for UARTA clock source | |
| Enumerator | |
| UARTA_CLOCK_SOURCE_SOSC_LOCO | SOSC/LOCO. |
| UARTA_CLOCK_SOURCE_LOCO | LOCO. |
| UARTA_CLOCK_SOURCE_MOSC | MOSC. |
| UARTA_CLOCK_SOURCE_HOCO | HOCO. |
| UARTA_CLOCK_SOURCE_MOCO | MOCO. |
| UARTA_CLOCK_SOURCE_SOSC | SOSC. |

### ◆ uarta_clock_div_t

| enum uarta_clock_div_t | |
|---|---|
| Enumeration for UARTA clock divider | |
| Enumerator | |
| UARTA_CLOCK_DIV_1 | fSEL/1 |
| UARTA_CLOCK_DIV_2 | fSEL/2 |
| UARTA_CLOCK_DIV_4 | fSEL/4 |
| UARTA_CLOCK_DIV_8 | fSEL/8 |
| UARTA_CLOCK_DIV_16 | fSEL/16 |
| UARTA_CLOCK_DIV_32 | fSEL/32 |
| UARTA_CLOCK_DIV_64 | fSEL/64 |
| UARTA_CLOCK_DIV_COUNT | Total number of clock divider options. |

### ◆ uarta_clock_out_t

| enum uarta_clock_out_t | |
|---|---|
| Enabled/Disabled Clock output | |
| Enumerator | |
| UARTA_CLOCK_OUTPUT_DISABLED | Disables CLKAn output. |
| UARTA_CLOCK_OUTPUT_ENABLED | Enables CLKAn output. |

### ◆ uarta_dir_bit_t

| enum uarta_dir_bit_t | |
|---|---|
| Transmission/reception order configuration. | |
| Enumerator | |
| UARTA_DIR_BIT_MSB_FIRST | MSB first. |
| UARTA_DIR_BIT_LSB_FIRST | LSB first. |

### ◆ uarta_alv_bit_t

| enum uarta_alv_bit_t | |
|---|---|
| Transmission/reception level configuration. | |
| Enumerator | |
| UARTA_ALV_BIT_POSITIVE_LOGIC | Positive logic (wait state = high level, start bit = low level, stop bit = high level) |
| UARTA_ALV_BIT_NEGATIVE_LOGIC | Negative logic (wait state = low level, start bit = high level, stop bit = low level) |

## Function Documentation

### ◆ R_UARTA_Open()

fsp_err_t R_UARTA_Open ( uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg* )

Configures the UARTA driver based on the input configurations. If transmission/reception is enabled at compile time, transmission/reception is enabled at the end of this function. Implements uart_api_t::open

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_INVALID_ARGUMENT | Invalid clock select (f_UTA0) and baudrate configuration. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

### ◆ R_UARTA_Close()

fsp_err_t R_UARTA_Close ( uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_UARTA_Read()

fsp_err_t R_UARTA_Read ( uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data reception successfully ends. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A previous read operation is still in progress. |
| FSP_ERR_UNSUPPORTED | UARTA_CFG_RX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_UARTA_Write()

fsp_err_t R_UARTA_Write ( uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes* )

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data transmission finished successfully. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL. Number of transfers outside the max or min boundary when transfer instance used |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_IN_USE | A UARTA transmission is in progress |
| FSP_ERR_UNSUPPORTED | UART_CFG_TX_ENABLE is set to 0 |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

### ◆ R_UARTA_CallbackSet()

fsp_err_t R_UARTA_CallbackSet ( uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void *const *p_context*, uart_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_UARTA_BaudSet()

fsp_err_t R_UARTA_BaudSet ( uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting* )

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a uarta_baud_setting_t structure. Implements uart_api_t::baudSet

**Warning**

This terminates any in-progress transmission.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL or the UART is not configured to use the internal clock. |
| FSP_ERR_INVALID_ARGUMENT | Invalid clock select (f_UTA0) and baudrate configuration. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_UARTA_InfoGet()

fsp_err_t R_UARTA_InfoGet ( uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info* )

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in provided p_info. |
| FSP_ERR_ASSERTION | Pointer to UART control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_UARTA_Abort()

fsp_err_t R_UARTA_Abort ( uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort* )

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UARTA transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::disable

### ◆ R_UARTA_ReadStop()

fsp_err_t R_UARTA_ReadStop ( uart_ctrl_t *const *p_api_ctrl*, uint32_t * *p_remaining_bytes* )

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UARTA transaction aborted successfully. |
| FSP_ERR_ASSERTION | Pointer to UARTA control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_UNSUPPORTED | The requested Abort direction is unsupported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::disable

### ◆ R_UARTA_BaudCalculate()

fsp_err_t R_UARTA_BaudCalculate ( uint32_t *baudrate*, uint32_t *baud_rate_percent_error_x1000*, uarta_clock_source_t *clock_source*, uarta_baud_setting_t *const *p_baud_setting* )

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

**Parameters**

| | | |
|---|---|---|
| [in] | baudrate | Baud rate [bps]. For example, 19200, 57600, 115200, etc. |
| [in] | baud_rate_percent_error_x1000 | Max baud rate error. At most baud_rate_percent_error x 1000 required for module to function. Absolute max baud_rate_error is 4740 (4.74%). |
| [in] | clock_source | Clock Source. Required for module to generate baudrate. The clock sources can be select include UARTA_CLOCK_SOURCE_MOSC, UARTA_CLOCK_SOURCE_HOCO, UARTA_CLOCK_SOURCE_MOCO, UARTA_CLOCK_SOURCE_SOSC_LOCO. |
| [out] | p_baud_setting | Baud setting information stored here if successful |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate is set successfully |
| FSP_ERR_ASSERTION | Null pointer |
| FSP_ERR_INVALID_ARGUMENT | Argument is out of available range, baud rate is '0'. |
| FSP_ERR_INVALID_RATE | Baud rate error is outside the range or the baud rate could not be set given the current clock source. |

### ◆ R_UARTA_ReceiveSuspend()

| fsp_err_t R_UARTA_ReceiveSuspend ( uart_ctrl_t *const *p_api_ctrl*) |
|---|
| Suspend Reception<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Functionality not supported by this driver instance |

### ◆ R_UARTA_ReceiveResume()

| fsp_err_t R_UARTA_ReceiveResume ( uart_ctrl_t *const *p_api_ctrl*) |
|---|
| Resume Reception<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Functionality not supported by this driver instance |

## 5.2.6.32 UART Communication Device (rm_comms_uart)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | RM_COMMS_UART_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg)<br>Opens and configures the UART Comms module. Implements rm_comms_api_t::open. More... |
| fsp_err_t | RM_COMMS_UART_Close (rm_comms_ctrl_t *const p_api_ctrl)<br>Disables specified UART Comms module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_UART_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)<br>Updates the UART Comms callback. Implements rm_comms_api_t::callbackSet. More... |

| | |
|---|---|
| fsp_err_t | RM_COMMS_UART_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)<br>Performs a read from the UART device. Implements rm_comms_api_t::read. More... |
| fsp_err_t | RM_COMMS_UART_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write to the UART device. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_UART_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs a write to, then a read from the UART device. Implements rm_comms_api_t::writeRead. More... |

### Detailed Description

Middleware to implement a generic communications interface over UART. This module implements the Communications Middleware Interface.

# Overview

The RM_COMMS_UART module implements COMMS API for UART interface.

### Features

The implementation of the UART communications interfacehas the following key features:

- Non-blocking API for bare metal
- Non-blocking and blocking API for RTOS

# Configuration

### Build Time Configurations for rm_comms_uart

The following build time configurations are defined in fsp_cfg/rm_comms_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Connectivity > UART Communication Device (rm_comms_uart)

This module can be added to the Stacks tab via New Stack > Connectivity > UART Communication Device (rm_comms_uart).

| Configuration | Options | Default | Description |
|---|---|---|---|
| RTOS | | | |
| Recursive Write Mutex | • Do Not Use<br>• Use | Use | Lock device for writing in using RTOS. |
| Recursive Read Mutex | • Do Not Use<br>• Use | Use | Lock device for reading in using RTOS. |
| Mutex Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for recursive mutex operation in using RTOS. |
| Write Semaphore | • Do Not Use<br>• Use | Use | Block writing in using RTOS. |
| Read Semaphore | • Do Not Use<br>• Use | Use | Block reading in using RTOS. |
| Semaphore Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for semaphore operation in using RTOS. |
| Name | Name must be a valid C symbol | g_comms_uart0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. |

# Usage Notes

### Limitations

- RM_COMMS_API are not reentrant in non blocking mode
- When in blocking mode, RM_COMMS_UART_Write() and RM_COMMS_UART_Read() cannot be called in callback.
- RM_COMMS_UART_WriteRead API is not implemented

# Examples

### Basic Example

This is a basic example of minimal use of UART communications implementation in an application.

```
void rm_comms_uart_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_COMMS_UART_Open(&g_comms_uart_ctrl, &g_comms_uart_cfg);
 if (FSP_SUCCESS != err)
    {
```

```
 /* Handle any errors. */
    }
 while (true)
    {
 /* Send data. */
       g_err_flag = 0;
       g_tx_flag = 0;
       err       = RM_COMMS_UART_Write(&g_comms_uart_ctrl, g_tx_buf, TX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors. */
      }
 while ((0 == g_tx_flag) && (0 == g_err_flag))
      {
 /* Wait callback */
      }
 /* Receive data. */
       g_err_flag = 0;
       g_rx_flag = 0;
       err       = RM_COMMS_UART_Read(&g_comms_uart_ctrl, g_rx_buf, RX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors.*/
      }
 while ((0 == g_rx_flag) && (0 == g_err_flag))
      {
 /* Wait callback */
      }
    }
}
static void rm_comms_uart_callback (rm_comms_callback_args_t * p_args)
{
 if (p_args->event == RM_COMMS_EVENT_TX_OPERATION_COMPLETE)
    {
```

```
        g_tx_flag = 1;
    }
 else if (p_args->event == RM_COMMS_EVENT_RX_OPERATION_COMPLETE)
    {
        g_rx_flag = 1;
    }
 else
    {
        g_err_flag = 1;
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_comms_uart_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_comms_uart_instance_ctrl_t

| struct rm_comms_uart_instance_ctrl_t | |
|---|---|
| Communications middleware control structure. | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_comms_cfg_t const * | p_cfg |
| | Middleware configuration. |
| | |
| rm_comms_uart_extended_cfg_t const * | p_extend |
| | Pointer to extended configuration structure. |
| | |
| void(* | p_callback )(rm_comms_callback_args_t *p_args) |
| | Pointer to callback that is called when a uart_event_t occurs. |

| | |
|---|---|
| void * | p_context |
| | Pointer to context passed into callback function. |

### Function Documentation

#### ◆ RM_COMMS_UART_Open()

fsp_err_t RM_COMMS_UART_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the UART Comms module. Implements rm_comms_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | UART Comms module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

#### ◆ RM_COMMS_UART_Close()

fsp_err_t RM_COMMS_UART_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified UART Comms module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_UART_CallbackSet()

fsp_err_t RM_COMMS_UART_CallbackSet ( rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context* )

Updates the UART Comms callback. Implements rm_comms_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_UART_Read()

fsp_err_t RM_COMMS_UART_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the UART device. Implements rm_comms_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_UART_Write()

fsp_err_t RM_COMMS_UART_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the UART device. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_UART_WriteRead()

fsp_err_t RM_COMMS_UART_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs a write to, then a read from the UART device. Implements rm_comms_api_t::writeRead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Not supported. |

## 5.2.6.33 USB (r_usb_basic)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | R_USB_EventGet (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)<br>Obtains completed USB related events. (OS-less Only) More... |
| fsp_err_t | R_USB_Callback (usb_callback_t *p_callback)<br>Register a callback function to be called upon completion of a USB related event. (RTOS only) More... |
| fsp_err_t | R_USB_Open (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg)<br>Applies power to the USB module specified in the argument (p_ctrl). More... |
| fsp_err_t | R_USB_Close (usb_ctrl_t *const p_api_ctrl)<br>Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module). More... |
| fsp_err_t | R_USB_Read (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)<br>Bulk/Interrupt data transfer. More... |
| fsp_err_t | R_USB_Write (usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination) |

| | |
|---|---|
| | Bulk/Interrupt data transfer. More... |
| fsp_err_t | R_USB_Stop (usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination)<br>Requests a data read/write transfer be terminated when a data read/write transfer is being performed. More... |
| fsp_err_t | R_USB_Suspend (usb_ctrl_t *const p_api_ctrl)<br>Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure. More... |
| fsp_err_t | R_USB_Resume (usb_ctrl_t *const p_api_ctrl)<br>Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure. More... |
| fsp_err_t | R_USB_VbusSet (usb_ctrl_t *const p_api_ctrl, uint16_t state)<br>Specifies starting or stopping the VBUS supply. More... |
| fsp_err_t | R_USB_InfoGet (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination)<br>Obtains completed USB-related events. More... |
| fsp_err_t | R_USB_PipeRead (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)<br>Requests a data read (Bulk/Interrupt transfer) via the pipe specified in the argument. More... |
| fsp_err_t | R_USB_PipeWrite (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)<br>Requests a data write (Bulk/Interrupt transfer). More... |
| fsp_err_t | R_USB_PipeStop (usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number)<br>Terminates a data read/write operation. More... |
| fsp_err_t | R_USB_UsedPipesGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination)<br>Gets the selected pipe number (number of the pipe that has |

| | |
|---|---|
| | completed initialization) via bit map information. More... |
| fsp_err_t | R_USB_PipeInfoGet (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)<br>Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. More... |
| fsp_err_t | R_USB_PullUp (usb_ctrl_t *const p_api_ctrl, uint8_t state)<br>This API enables or disables pull-up of D+/D- line. More... |
| fsp_err_t | R_USB_HostControlTransfer (usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)<br>Performs settings and transmission processing when transmitting a setup packet. More... |
| fsp_err_t | R_USB_PeriControlDataGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)<br>Receives data sent by control transfer. More... |
| fsp_err_t | R_USB_PeriControlDataSet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)<br>Performs transfer processing for control transfer. More... |
| fsp_err_t | R_USB_PeriControlStatusSet (usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status)<br>Set the response to the setup packet. More... |
| fsp_err_t | R_USB_RemoteWakeup (usb_ctrl_t *const p_api_ctrl)<br>Sends a remote wake-up signal to the connected Host. More... |
| fsp_err_t | R_USB_DriverActivate (usb_ctrl_t *const p_api_ctrl)<br>Activate USB Driver for USB Peripheral BareMetal. More... |
| fsp_err_t | R_USB_CallbackMemorySet (usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory)<br>Set callback memory to USB Driver for USB Peripheral BareMetal. More... |

| | |
|---|---|
| fsp_err_t | R_USB_ModuleNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)<br>This API gets the module number. More... |
| fsp_err_t | R_USB_ClassTypeGet (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)<br>This API gets the class type. More... |
| fsp_err_t | R_USB_DeviceAddressGet (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)<br>This API gets the device address. More... |
| fsp_err_t | R_USB_PipeNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)<br>This API gets the pipe number. More... |
| fsp_err_t | R_USB_DeviceStateGet (usb_ctrl_t *const p_api_ctrl, uint16_t *state)<br>This API gets the state of the device. More... |
| fsp_err_t | R_USB_DataSizeGet (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)<br>This API gets the read data size. More... |
| fsp_err_t | R_USB_SetupGet (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)<br>This API gets the setup information. More... |
| fsp_err_t | R_USB_OtgCallbackSet (usb_ctrl_t *const p_api_ctrl, usb_otg_callback_t *p_callback)<br>Set callback function to be called when the OTG role swap was completed on Azure RTOS. More... |
| fsp_err_t | R_USB_OtgSRP (usb_ctrl_t *const p_api_ctrl)<br>Start the SRP processing for OTG on Azure RTOS. More... |

## Detailed Description

Driver for the USB peripheral on RA MCUs. This module implements the USB Interface.

# Overview

The USB module operates in combination with the device class drivers provided by Renesas to form a complete USB stack.

### Features

The USB module has the following key features:

- USB Host mode
  - Enumerates Low/Full/High-speed devices (see note below)
  - Automatic transfer error determination and retry
- USB Peripheral mode
  - Supports USB1.1/2.0/3.0 hosts
- Automatic processing of device connect/disconnect, suspend/resume, and USB bus reset
- Up to 10 pipes
  - Control transfers supported on pipe 0
  - Data transfer on pipes 1 to 9 (Bulk or Interrupt)
- Functions with or without an RTOS

*Note*

*Supported speeds are dependent on the MCU.*

# Support Device Class

This driver supports the following device classes.

| Host/Peripheral | Device Class | BareMetal | FreeRTOS | AzureRTOS |
|---|---|---|---|---|
| Host | HCDC (ACM) | Yes | Yes | Yes |
| | HCDC (ECM) | – | Yes | – |
| | HHID | Yes | Yes | Yes |
| | HMSC | Yes | Yes | Yes |
| | HPRN | – | – | Yes |
| | HUVC | – | – | Yes |
| | HVND | Yes | Yes | – |
| | HAUD | Yes | Yes | Yes |
| | HCDC+HMSC (Composite) | – | Yes | – |
| Peripheral | PCDC | Yes | Yes | Yes |
| | PHID | Yes | Yes | Yes |
| | PMSC | Yes | Yes | Yes |
| | PPRN | Yes | Yes | Yes |
| | PAUD | Yes | Yes | Yes |
| | DFU | – | – | Yes |

| | | | | |
|---|---|---|---|---|
| | PVND | Yes | Yes | – |
| | PCDC+PMSC | Yes | Yes | Yes |
| | PHID+PMSC | Yes | Yes | – |
| | PCDC+PHID | Yes | Yes | – |
| | PCDC+PCDC | Yes | Yes | – |
| | PHID+PHID | Yes | Yes | – |
| | PCDC+PVND | Yes | Yes | – |
| Other | OTG (CDC) | – | – | Yes |
| | OTG (HID) | – | – | Yes |
| | OTG (MSC) | – | – | Yes |

# Configuration

### Build Time Configurations for r_usb_basic

The following build time configurations are defined in fsp_cfg/r_usb_basic_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| CL Only Mode | • Disabled<br>• Enabled | Disabled | To reduce power consumption, a classic-only mode where operation is only in accord with the USB 1.1 standard is selectable. |
| CPU Bus Access Wait Cycles | MCU Specific Options | | This setting controls the delay for consecutive USB peripheral register access. Set this value to a number of CPU cycles that is equivalent to 40.8ns or more. |
| Battery Charging | MCU Specific Options | | Specify whether or not to include battery charging functionality. |
| Power IC Shutdown Polarity | MCU Specific Options | | Select the polarity of the Shutdown signal on the power supply IC (if provided). |

| | | | |
|---|---|---|---|
| Dedicated Charging Port (DCP) Mode | MCU Specific Options | | When enabled, USB communication is disabled and the port is used for charging only. |
| Notifications for SET_INTERFACE/SET_FEATURE/CLEAR_FEATURE | • Disabled<br>• Enabled | Enabled | When enabled, the application will receive notifications for SET_INTERFACE, SET_FEATURE and CLEAR_FEATURE messages. |
| Double Buffering | • Disabled<br>• Enabled | Enabled | When enabled, the FIFOs for Pipes 1-5 are double-buffered. |
| Continuous Transfer Mode | MCU Specific Options | | Enable or disable continuous transfer mode. |
| LDO Regulator | MCU Specific Options | | Enable or disable LDO regulator. |
| DMA/DTC Support | MCU Specific Options | | Enable or disable DMA/DTC support for the USB module. |
| DMA/DTC Source Address | MCU Specific Options | | Set this to match the speed mode when DMA/DTC is enabled. Otherwise, set to 'DMA Disabled'/'DTC Disabled'. |
| DMA/DTC Destination Address | MCU Specific Options | | Set this to match the speed mode when DMA/DTC is enabled. Otherwise, set to 'DMA Disabled'/'DTC Disabled'. |
| USB Compliance Test mode | • Enabled<br>• Disabled | Disabled | Display the information required to take the compliance test. |
| USB TPL table name | Enter the TPL table name. | NULL | Enter the name of the TPL Table. |

**Configurations for Connectivity > USB (r_usb_basic)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_basic0 | Module name. |
| USB Mode | • Host mode<br>• Peri mode<br>• OTG mode | Host mode | Select the usb mode. |

| | | | |
|---|---|---|---|
| USB Speed | • Full Speed<br>• Hi Speed<br>• Low Speed | Full Speed | Select the USB speed. |
| USB Module Number | • USB_IP0 Port<br>• USB_IP1 Port | USB_IP0 Port | Specify the USB module number to be used. |
| USB Device Class | Refer to the RA Configuration tool for available options. | Peripheral Communications Device Class | Select the USB device class. |
| USB Descriptor | USB Descriptor must be a valid C symbol. | g_usb_descriptor | Enter the name of the descriptor to be used. For how to create a descriptor structure, refer to the Descriptor definition chapter in the usb_basic manual. Specify NULL when using the Host class. |
| USB Compliance Callback | Compliance Callback must be a valid C symbol. | NULL | Set the callback for compliance tests here. |
| USBFS Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the main USBFS ISR. |
| USBFS Resume Priority | MCU Specific Options | | Select the interrupt priority used by the USBFS Resume ISR. |
| USBFS D0FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the USBFS D0FIFO. |
| USBFS D1FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the USBFS D1FIFO. |
| USBHS Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the main USBHS ISR. |
| USBHS D0FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the USBHS D0FIFO ISR. |
| USBHS D1FIFO Interrupt Priority | MCU Specific Options | | Select the interrupt priority used by the USBHS D1FIFO ISR. |
| USB Callback | Enter the address of the function. | NULL | A user callback function can be defined here. |
| USB Callback Context | Enter the address of the context. | NULL | Set the callback context here. |

### Clock Configuration

The USB module uses PLL as the clock source. The PLL frequency can be set in the **Clocks** tab of the configuration editor or by using the CGC Interface at run-time.

*Note*

*When using HOCO as the PLL source on Cortex M33 parts the FLL function must be enabled for correct USB operation. Refer to the MCU Family -> Clocks group of the BSP properties in the RA configuration tool to adjust FLL settings.*

### Pin Configuration

In peripheral mode the USB_VBUS and/or USBHS_VBUS pins are used to detect the USB connection status (connected or disconnected) and should be connected to the USB VBUS signal.

*Note*

*USB_VBUS and USBHS_VBUS are 5V-tolerant pins.*

In host mode the USBHS_VBUSEN, USBHS_OVRCURA and USBHS_OVRCURB pins should be connected to the relevant pins of an external power supply IC, if available. These pins will be used to manage the USB VBUS supply.

### DMA Configuration

When using DMA with USB the following properties must be configured for each DMAC module:

| Config Name | Select Name | Description |
|---|---|---|
| Transfer Size | 2 Bytes<br>4 Bytes | In FS mode, select "2 Bytes"<br>In HS mode, select "4 Bytes" |
| Activation source | USBFS FIFO 0<br>USBFS FIFO 1<br>USBHS FIFO 0<br>USBHS FIFO 1 | USB FS Reception<br>USB FS Transmission<br>USB HS Reception<br>USB HS Transmission |

# Descriptor definition

In Peripheral mode, the usb_descriptor_t structure stores descriptor information including the device and configuration descriptors. The values set in this structure are sent to the USB host as part of enumeration.

```
typedef struct usb_descriptor
{
    uint8_t *p_device;    /* Pointer to device descriptor */
    uint8_t *p_config_f; /* Pointer to full-speed configuration descriptor */
    uint8_t *p_config_h; /* Pointer to high-speed configuration descriptor (HS only)
*/
    uint8_t *p_qualifier; /* Pointer to device qualifier descriptor (HS only) */
```

```
    uint8_t **pp_string; /* Pointer to string descriptor table */
    uint8_t num_string;   /* Number of strings in table */
} usb_descriptor_t;
```

*Note*

*Even in high-speed mode the full-speed configuration must be made available:*

```
/* Example USB FS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
    smp_config_f,
    NULL,
    NULL,
    smp_str_table,
    3,
};
/* Example USB HS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
    smp_config_f,
    smp_config_h,
    smp_qualifier,
    smp_str_table,
    3,
};
```

### String Descriptor

This USB driver requires string descriptors to be registered in the string descriptor table. Use the following format to define the elements:

```
/* String descriptor 0 is reserved for language ID information */
uint8_t str_descriptor_0[]
{
```

```
    0x04,      /* Length */
    0x03,      /* Descriptor type */
    0x09, 0x04 /* Language ID */
};
uint8_t str_descriptor_manufacturer[] =
{
    0x10,      /* Length */
    0x03,      /* Descriptor type */
 'R', 0x00,
 'E', 0x00,
 'N', 0x00,
 'E', 0x00,
 'S', 0x00,
 'A', 0x00,
 'S', 0x00
};
uint8_t str_descriptor_product[] =
{
    0x12,      /* Length */
    0x03,      /* Descriptor type */
 'C', 0x00,
 'D', 0x00,
 'C', 0x00,
 '_', 0x00,
 'D', 0x00,
 'E', 0x00,
 'M', 0x00,
 'O', 0x00
};
/* String descriptor table */
uint8_t * smp_str_table[] =
{
    str_descriptor_0,            /* Index: 0 */
    str_descriptor_manufacturer, /* Index: 1 */
```

```
    str_descriptor_product,      /* Index: 2 */
};
```

*Note*

*Set the string index values in the device/configuration descriptors (iManufacturer, iConfiguration etc.) to the index of the desired string in the string descriptor table. For example, in the table below, the manufacturer is described in* ***str_descriptor_manufacturer*** *and the value of iManufacturer in the device descriptor is* ***1****.*

### Other Descriptors

Refer to the Universal Serial Bus Revision 2.0 specification (http://www.usb.org/developers/docs/) for details on how to construct the device, configuration and qualifier descriptors.

# Usage Notes

### Program Structure

USB applications (whether using an RTOS or not) should be written as an event-handling loop. Either a callback function (RTOS only) or R_USB_EventGet should be used to provide event data to the application loop where a switch statement handles the event.

*Note*

*1.The USB_STATUS_CONFIGURED event should be confirmed before calling R_USB_Read or R_USB_Write.*
*2.When attaching or detaching USB cable, the suspend or resume event may be notified to the application program in USB peripheral mode. Please ignore these events since the notification of these events to the application program does not affect the operation.*
*3.USB applications always need to check the USB device status member(status) of usb_event_info_t control structure before proceeding to event-handling loop. If device status is success (FSP_SUCCESS) then only proceed with handling of received events.*

### Limitations

Developers should be aware of the following limitations when using the USB driver:

- In USB host mode, the module does not support suspend during data transfers. Execute suspend only after confirming that all transfers are complete.
- Multiconfigurations are not supported.
- This driver does not support CPU transfers using the D0FIFO/D1FIFO register.
- In USB host mode, this USB driver does not support the composite device other than "HMSC + HCDC (for FreeRTOS)".
- The user can not specify DMA transfer to USB IP0 and IP1 modules at the same time when using USB multi-port feature. USB multi-port function: Simultaneous operation feature of USB Host and Peripheral.
- In USB host mode, When using a USB High-speed module, enumeration may not start when the user slowly detach the USB cable and attach USB cable.

### Compliance Test

Please set as follows to the following items in RA configuration (r_usb_basic) when doing the compliance test.

1.USB Compliance Test mode Set "Enabled" in this item.

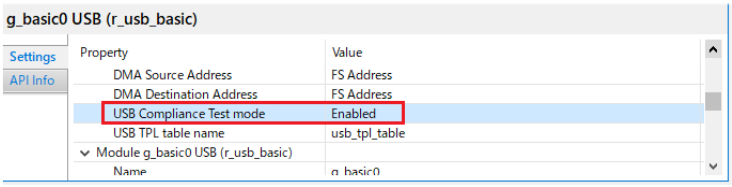



Figure 223: Compliance Test Setting

2.USB TPL table name. Set the start address of TPL(Target Peripheral List) defined in the application program.

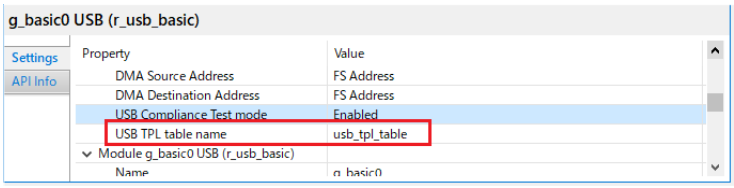



Figure 224: TPL Start Address Setting

Please refer to the following about how to define for TPL.
The following example is when two devices are set in TPL.

```
const uint16_t usb_tpl_table[] =
{
    2,                  /* Number of tpl */
    0,                  /* Reserved */
    0x0123, 0x4567,     /* Vendor ID, Product ID (1st device) */
    0x89ab, 0xcdef      /* Vendor ID, Product ID (2nd device) */
};
```

3.USB Compliance Callback Set the start address of the callback function defined in the application program.

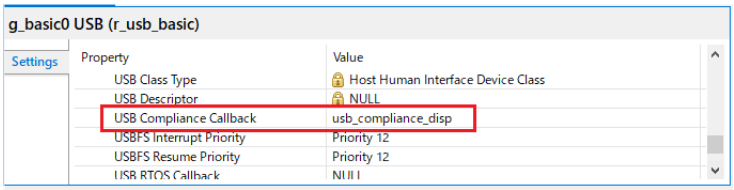



Figure 225: Compliance Callback Setting

The user needs to create this callback function by referring to the following.

```
void usb_compliance_disp (void * param)
{
    usb_compliance_t    *disp_data;
    uint8_t             print_data[32];
    disp_data = (usb_compliance_t*)param;
 switch(disp_data->status)
    {
 case USB_COMPLIANCETEST_ATTACH:            /* Device Attach Detection */
           display("\nATTACH \n");
 break;
 case USB_COMPLIANCETEST_DETACH:            /* Device Detach Detection */
           display("\nDETACH \n");
 break;
 case USB_COMPLIANCETEST_TPL:               /* TPL device connect */
           display("\nTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_NOTTPL:            /* Not TPL device connect */
           display("\nNOTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_HUB:               /* USB Hub connect */
           display("\nHub    \n");
 break;
 case USB_COMPLIANCETEST_OVRC:             /* over current */
 break;
 case USB_COMPLIANCETEST_NORES:             /* Response Time out for Control Read
Transfer */
           display("\nNOTRESP \n");
 break;
 case USB_COMPLIANCETEST_SETUP_ERR:         /* Setup Transaction Error */
 break;
 default:
 break;
    }
} /* End of function usb_compliance_disp() */
```

Please replace the display function described in the example with the display function created by the customer.

### Callback function for USB Peripheral

The user can get completed USB event using callback function in USB Peripheral for BareMetal.

- Please specify user callback function in the following item.

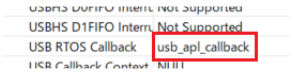



Figure 226: User Callback Function Setting

- Please call the *R_USB_DriverActivate* function using one of the following methods.
  - Infinite loop in user application program
  - Timer Interrupt (Add r_gpt module in user system)
- Please sepecify the area to store USB events to USB driver using *R_USB_CallbackMemorySet* function after calling *R_USB_Open* function.

```
usb_callback_args_t g_apl_usb_event_callback;
```

- Please allocate a buffer area such as a ring buffer to store USB event information in user application program.

```
#define BUFSIZE 10

usb_callback_args_t g_apl_usb_event_buffer[BUFSIZE];
```

- The USB driver calls the callback function when the USB event completes. USB event information is stored in the area indicated by the callback function argument.Copy the callback function argument to the ring buffer area or other area secured above.

```
g_apl_usb_event_buffer[g_apl_buffer_wp] = g_apl_usb_event_callback;

g_apl_buffer_wp++;

g_apl_buffer_wp %= BUFSIZE;
```

### TrustZone

1. The USB driver for FreeRTOS cannot be allocated in Secure region.
2. Please place the descriptor file in Secure region when using Non-Secure Callable.
3. The user callback function should be specified using the *R_USB_Callback* function after calling the *R_USB_Open* function when using Non-Secure Callable.
4. For Non-Secure region, please do USB clock setting when creating Secure project.
5. Please do USB pin setting when creating USB driver project.

### Support Composite Device

This driver for FreeRTOS supports the following compoiste device when this driver for FreeRTOS works in USB Host mode.

1. PCDC + PMSC

### How to Configuration

- Add HMSC and HCDC stack as follow.

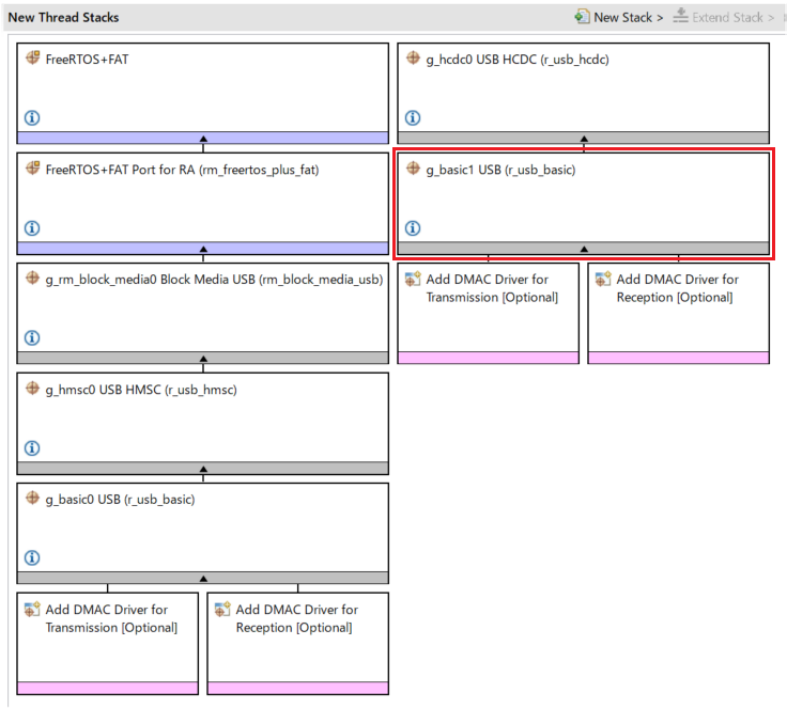



Figure 227: Add HMSC and HCDC Stack

- Delete the "g_basic1" instance manually since this instance is not used in composite device. (Refer to the red frame in the above figure.)

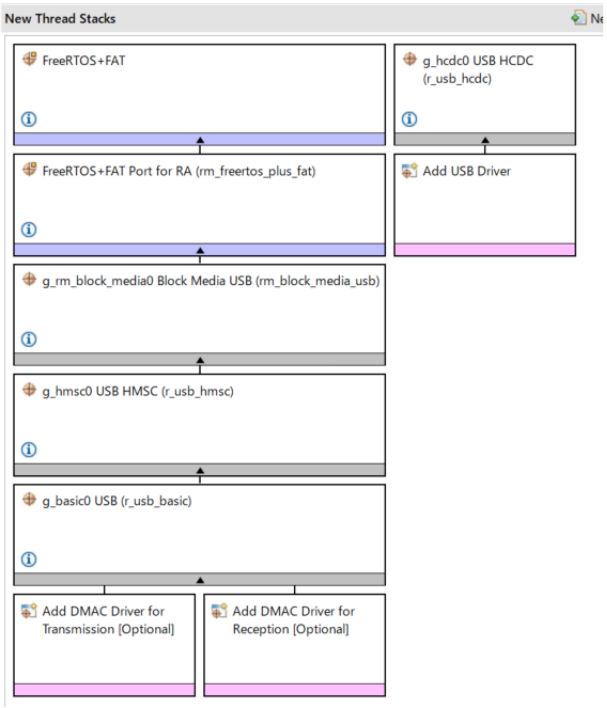



Figure 228: Delete g_basic1 instance

### Using Hub

- The user can use USB Hub when using the following device classes (Bare Metal + FreeRTOS):
    - Host Communication Device Class (HCDC)
    - Host Human Interface Device Class (HHID)
    - Host Mass Storage Class (HMSC)
- When configuring single-class host hub, be sure to enable hub support property in each class stack:

Figure 229: USB Hub CDC-CDC setting

#### Note

- For USB Hub usage with Azure RTOS, please refer to Azure RTOS USBX Port (rm_usbx_port).
- The followings are the limitation when using USB Hub:
    - The current USB driver does not support for multi-class host hub and DMA transfer.
    - The USB driver allocates USB PIPE9 for USB Hub.
    - For HCDC, the user cannot connect more than 2 CDC devices to USB Hub.
    - For HHID, the user cannot connect more than 3 HID devices to USB Hub.
    - For HMSC, the user cannot connect more than 4 MSC devices to USB Hub.
- Depending on the Hub used, it may be necessary to supply power to the Hub.
- For HHID Hub, do not connect two HID devices that support both Interrupt IN and Interrupt Out transfer.
- For HMSC Hub, HMSC driver may not work properly due to the combination of the Hub

device and USB stick.
- HVND and HCDC-ECM do not support USB Hub.

### UCLK setting

Enable UCLK in "Clocks" tab on e² studio when using the following MCU.

1. RA6M4

# Examples

### USB Basic Example

This is a basic example of minimal use of the USB in an application.

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    memset(&event_info, 0, sizeof(usb_event_info_t));
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
     {
 case USB_STATUS_CONFIGURED:
               g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (FSP_SUCCESS == event_info.status)
     {
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
     }
 break;
 case USB_STATUS_READ_COMPLETE:
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
```

```
      {
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
      }
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else
      {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
      }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
      }
    }
} /* End of function usb_main() */
```

### USB Host Composite (CDC+MSC) Example

This is Host composiete (CDC+MSC) example of minimal use of the USB in an application.

```
#define RESET_VALUE (0x00)
#define EP_INFO \
  "\r\nThis example project demonstrates basic functionalities of USB Host
Communication Device \r\n" \
  "Class (HCDC) driver on Renesas RA MCUs using 2 RA Boards. The Board 1(with USB
HCDC Example \r\n" \
  "running on it)communicates with Board2(with USB PCDC Example project running).
Board 1 initiates\r\n" \
  "the communication by sending commands to Board 2 and Board 2 responds by sending
the data.\r\n" \
  "Board1 prints the received data on the RTTViewer.\r\n\n\n"
#define HCDC_TSK_STACK_SIZE 1024
#define HCDC_TSK_PRI 2
#define BUF_SIZE 512
#define KIT_INFO ('1')
#define NEXT_STEPS ('2')
#define CARRIAGE_RETURN ('\r')
#define SET_LINE_CODING (USB_CDC_SET_LINE_CODING | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define GET_LINE_CODING (USB_CDC_GET_LINE_CODING | USB_DEV_TO_HOST | USB_CLASS |
USB_INTERFACE)
#define SET_CONTROL_LINE_STATE (USB_CDC_SET_CONTROL_LINE_STATE | USB_HOST_TO_DEV |
USB_CLASS | USB_INTERFACE)
#define LINE_CODING_LENGTH (0x07U)
#define VALUE_ZERO (0x0000U)
#define NO_WAIT_TIME 0
#define CDC_READ_DATA_LEN 512
#define CDC_WRITE_DATA_LEN 512
#define ZERO_INDEX 0
void usb_app_hcdc_task(void * pvParameters);
static void usb_app_common_callback(usb_event_info_t * p_event_info, usb_hdl_t
cur_task, usb_onoff_t usb_state);
static void usb_app_hcdc_callback(usb_event_info_t * p_event_info, usb_hdl_t
cur_task, usb_onoff_t usb_state);
```

```
static void set_line_coding(usb_instance_ctrl_t * p_ctrl, uint8_t device_address);
static void set_control_line_state(usb_instance_ctrl_t * p_ctrl, uint8_t
device_address);
static void get_line_coding(usb_instance_ctrl_t * p_ctrl, uint8_t device_address);
static void usb_data_process(usb_event_info_t * event_info);
static void handle_error(fsp_err_t err, char * err_str);
static void process_usb_operation(uint8_t p_input_buffer);
static void usb_write_operation(void);
static void usb_read_operation(void);
static void format_usb_device(void);
static bool check_usb_connection(void);
static void usb_safely_eject(void);
static void update_buffer(void);
static void fat_clean_up(void);
static fsp_err_t usb_init(void);
static TaskHandle_t g_app_hcdc_tsk_hdl;
static usb_callback_t * g_usb_host_apl_callback[2];
static FF_Disk_t my_disk;
static usb_hcdc_linecoding_t g_serial_state;
static usb_hcdc_linecoding_t g_com_parm;
static uint8_t g_write_data[WRITE_ITEM_SIZE] = {RESET_VALUE}; /* Data(10k) to write
to file */
static uint8_t g_read_data[WRITE_ITEM_SIZE] = {RESET_VALUE}; /* Variable to store the
data read from file */
static uint8_t g_snd_buf[BUF_SIZE] BSP_ALIGN_VARIABLE(4) = {RESET_VALUE};
static uint8_t g_rcv_buf[BUF_SIZE] BSP_ALIGN_VARIABLE(4) = {RESET_VALUE};
static uint8_t g_usb_dummy = RESET_VALUE; /* dummy variable to send */
static volatile bool g_err_flag = false; /* error flag bit */
static bool b_usb_hmsc_close = false;
static volatile bool g_rm_freertos_plus_fat_insertion_events = false;
extern
void new_thread0_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
```

```
   BaseType_t err_task = pdFALSE;
   memset(&my_disk, RESET_VALUE, sizeof(my_disk));
 fsp_pack_version_t version = {RESET_VALUE};
 /* version get API for FLEX pack information */
 R_FSP_VersionGet(&version);
   g_usb_host_apl_callback[0] = usb_app_hcdc_callback;          // HCDC App Callback
   g_usb_host_apl_callback[1] = g_basic0_cfg.p_usb_apl_callback; // HMSC App
Callback
 R_USB_Callback(usb_app_common_callback);
 /* Example Project information printed on the Console */
   APP_PRINT(BANNER_INFO,
             EP_VERSION,
             version.version_id_b.major,
             version.version_id_b.minor,
             version.version_id_b.patch);
   APP_PRINT(EP_INFO);
 fsp_err_t freertos_fat_error = FSP_SUCCESS;
   freertos_fat_error = usb_init();
 if (FSP_SUCCESS != freertos_fat_error)
   {
      APP_ERR_PRINT("\r\nError in initializing FreeRTOS+FAT with USB_HMSC\r\n");
      APP_ERR_TRAP(freertos_fat_error);
   }
   err_task = xTaskCreate((TaskFunction_t) usb_app_hcdc_task,
 "HCDC_TSK",
                          HCDC_TSK_STACK_SIZE,
     NULL,
                          HCDC_TSK_PRI,
                          &g_app_hcdc_tsk_hdl);
 if (pdPASS != err_task)
   {
      APP_ERR_PRINT("\r\nAppTask Create failed.\r\n");
 return;
   }
```

```
 /* Print USB HMSC menu */
    APP_PRINT(USB_HMSC_MENU);
 while (true)
    {
 static uint8_t rtt_input_buf[BUFFER_SIZE_DOWN] = {RESET_VALUE};
 static uint8_t converted_rtt_input             = RESET_VALUE;
 /*Read RTT input from user*/
 if (APP_CHECK_DATA)
      {
            APP_READ(rtt_input_buf);
            converted_rtt_input = (uint8_t) atoi((char *) rtt_input_buf);
            process_usb_operation(converted_rtt_input);
      }
 if ((false == b_usb_hmsc_close) && (false ==
g_rm_freertos_plus_fat_insertion_events))
      {
            APP_PRINT("\r\n\n USB Device disconnected without Eject option.\r\n");
            APP_PRINT("\r\n Connect the USB and Execute Safely Eject option to make
sure file operations works"
 "correctly\r\n");
 /* Wait until USB Device is connected */
 while (true != check_usb_connection())
      {
                ;
      }
      }
       vTaskDelay(1);
    }
}
/***********************************************************************************
*******************************
 * This function initiates the USB operation by calling respective functions.
 ***********************************************************************************
*********************************/
```

```
static void process_usb_operation (uint8_t input_buffer)
{
 fsp_err_t freertos_fat_error = FSP_SUCCESS;
 switch (input_buffer)
    {
 case USB_WRITE:
      {
            usb_write_operation();
            usb_read_operation();
      APP_PRINT(USB_HMSC_MENU);
 break;
      }
 case USB_FORMAT:
      {
            format_usb_device();
      APP_PRINT(USB_HMSC_MENU);
 break;
      }
 case USB_SAFELY_EJECT:
      {
            usb_safely_eject();
      APP_PRINT(USB_HMSC_MENU);
 break;
      }
 case USB_INIT:
      {
 if (true == b_usb_hmsc_close)
      {
      freertos_fat_error = usb_init();
 if (FSP_SUCCESS != freertos_fat_error)
      {
         APP_ERR_PRINT("\r\nError in initializing FreeRTOS+FAT with USB_HMSC\r\n");
         APP_ERR_TRAP(freertos_fat_error);
      }
```

```
      }
 else
      {
                APP_PRINT("\r\n FreeRTOS+FAT USB_HMSC driver is already Initialized.
Provide any other command\r\n");
      }
      APP_PRINT(USB_HMSC_MENU);
 break;
      }
 default:
      {
            APP_PRINT("\r\n Invalid input. Provide a valid input\r\n");
 break;
      }
    }
}
/***********************************************************************************
*****************************
 * This function performs USB HMSC write operation.
 ***********************************************************************************
*****************************/
static void usb_write_operation (void)
{
    FF_FILE * file_pointer = NULL;
    FF_Stat_t file_details;
    int32_t   file_error = SUCCESS;
 /* Capture the number of bytes written in the variable to check write status. */
 size_t bytes_written = RESET_VALUE;
    memset(&file_details, RESET_VALUE, sizeof(file_details));
 /* Double check the connection again to ensure the USB device is still mounted */
 if ((true == check_usb_connection()) && (true != b_usb_hmsc_close))
    {
 /* Once connection is detected open file is write mode */
       file_pointer = ff_fopen((const char *) FILE_NAME, WRITE_MODE);
```

```
if (NULL != file_pointer)
     {
/* Fill write buffer with data */
           update_buffer();
           APP_PRINT("   USB write operation will be initiated.\n");
/* Write API writes */
           bytes_written = ff_fwrite(g_write_data, sizeof
(g_write_data[RESET_VALUE]), WRITE_ITEM_SIZE, file_pointer);
if (WRITE_ITEM_SIZE != bytes_written)
     {
              APP_ERR_PRINT(" ff_write API failed. Closing opened file.\r\n");
/* Adding extra %d as parses string and prints %d as-is. */
              APP_PRINT(" %d\r\n", stdioGET_ERRNO());
              file_error = ff_fclose(file_pointer);
if (SUCCESS != file_error)
     {
                APP_ERR_PRINT("ff_fclose API failed");
/* Adding extra %d as parses string and prints %d as-is. */
               APP_PRINT(" %d\r\n", stdioGET_ERRNO());
     }
return;
     }
/* Close the file after write operation and open again in read mode */
           file_error = ff_fclose(file_pointer);
if (SUCCESS != file_error)
     {
              APP_ERR_PRINT("ff_fclose API failed");
/* Adding extra %d as parses string and prints %d as-is. */
              APP_PRINT(" %d\r\n", stdioGET_ERRNO());
return;
     }
           file_error = ff_stat((const char *) FILE_NAME, &file_details);
/* ff_stat returns 0 on success and -1 on error */
if (SUCCESS == file_error)
```

```
     {
 /* Compare the actual bytes written and file size after write operation */
 if (bytes_written == file_details.st_size)
     {
                 APP_PRINT("   %d bytes Data successfully written to file %s \n",
bytes_written, FILE_NAME);
                 APP_PRINT("   Write operation is Successful \n");
     }
 else
     {
                 APP_ERR_PRINT("ff_write API failed ");
 /* Adding extra %d as parses string and prints %d as-is. */
                APP_PRINT(" %d\r\n", stdioGET_ERRNO());
 return;
     }
     }
 else
     {
              APP_ERR_PRINT("ff_stat API failed");
 /* Adding extra %d as parses string and prints %d as-is. */
              APP_PRINT(" %d\r\n", stdioGET_ERRNO());
 return;
     }
     }
 else
     {
           APP_ERR_PRINT("ff_fopen API failed");
 /* Adding extra %d as parses string and prints %d as-is. */
           APP_PRINT(" %d\r\n", stdioGET_ERRNO());
 return;
     }
   }
 else
   {
```

```
       APP_PRINT("USB Device disconnected or not initialized after Eject command\n");
    }
}
/***********************************************************************************************
*****************************
 * This function performs USB HMSC read operation.
 ***********************************************************************************************
*****************************/
static void usb_read_operation (void)
{
    FF_FILE * file_pointer = NULL;
    FF_Stat_t file_details;
    int32_t   file_error = SUCCESS;
    memset(&file_details, RESET_VALUE, sizeof(file_details));
 /* Double check the connection again to ensure the USB device is still mounted */
 if ((true == check_usb_connection()) && (true != b_usb_hmsc_close))
    {
 /* Open the file read mode to read data written previously */
       file_pointer = ff_fopen((const char *) FILE_NAME, READ_MODE);
 if (file_pointer != NULL)
     {
           APP_PRINT("   USB read operation will be initiated.\n");
 /* Capture the number of bytes read in the variable to check read status. */
 size_t bytes_read = RESET_VALUE;
           bytes_read = ff_fread(g_read_data, sizeof(g_read_data[RESET_VALUE]),
WRITE_ITEM_SIZE, file_pointer);
 if (READ_WRITE_FAILURE == bytes_read)
     {
               APP_ERR_PRINT(" ff_read API failed. Closing opened file \r\n",
stdioGET_ERRNO());
               file_error = ff_fclose(file_pointer);
 if (SUCCESS != file_error)
     {
                  APP_ERR_PRINT("ff_fclose API failed");
```

```
/* Adding extra %d as parses string and prints %d as-is. */
                APP_PRINT(" %d\r\n", stdioGET_ERRNO());
return;
    }
return;
    }
          file_error = ff_fclose(file_pointer);
if (SUCCESS != file_error)
    {
              APP_ERR_PRINT("ff_fclose API failed");
/* Adding extra %d as parses string and prints %d as-is. */
              APP_PRINT(" %d\r\n", stdioGET_ERRNO());
return;
    }
          file_error = ff_stat((const char *) FILE_NAME, &file_details);
/* ff_stat returns 0 on success and -1 on error */
if (SUCCESS == file_error)
    {
/* Compare the actual bytes written and file size after write operation */
if (bytes_read == file_details.st_size)
    {
                APP_PRINT("   %d bytes Data successfully read from file %s \n",
bytes_read, FILE_NAME);
                APP_PRINT("   Read operation is Successful \n");
    }
else
    {
                APP_ERR_PRINT("ff_write API failed ");
/* Adding extra %d as parses string and prints %d as-is. */
                APP_PRINT(" %d\r\n", stdioGET_ERRNO());
return;
    }
/* Compare Write and Read data. */
if (SUCCESS == memcmp(g_write_data, g_read_data, WRITE_ITEM_SIZE))
```

```
     {
                  APP_PRINT("\r\nWrite and Read data is same\r\n");
     }
 else
     {
                  APP_ERR_PRINT("\r\nWrite and Read data did not match\r\n");
     }
     }
 else
     {
               APP_ERR_PRINT("ff_stat API failed");
 /* Adding extra %d as parses string and prints %d as-is. */
               APP_PRINT(" %d\r\n", stdioGET_ERRNO());
 return;
     }
     }
 else
     {
           APP_ERR_PRINT("ff_fopen API failed");
 /* Adding extra %d as parses string and prints %d as-is. */
           APP_PRINT(" %d\r\n", stdioGET_ERRNO());
 return;
     }
   }
 else
   {
      APP_PRINT("USB Device disconnected or not initialized after Eject command\n");
   }
}
/*******************************************************************************
*****************************
 * This function performs USB HMSC format operation.
 *******************************************************************************
*****************************/
```

```
static void format_usb_device (void)
{
 if ((true == check_usb_connection()) && (true != b_usb_hmsc_close))
    {
       APP_PRINT("\r\n USB Device Formatting will be initiated. Formatting will take
time "
 "depending on USB Device capacity.\r\n");
       APP_PRINT(" Do not disconnect the USB device while formatting is in
progress.   Please Wait ...\r\n");
 /* Formatting time varies with USB Device capacity. Might take longer time for
higher capacity USB Device */
       FF_Error_t ff_error = FF_Format(&my_disk, my_disk.xStatus.bPartitionNumber,
pdFALSE, pdFALSE);
 if (FF_ERR_NONE != ff_error)
      {
            APP_ERR_PRINT("\r\n FF_Format API failed %d. Check the USB Device.\r\n",
FF_GetErrMessage(ff_error));
            APP_PRINT(" %d\r\n", FF_GetErrMessage(ff_error));
      }
 else
      {
            APP_PRINT("\r\nUSB Device Formatted successfully \r\n");
      }
    }
 else
    {
       APP_PRINT("USB Device disconnected or not initialized after Eject command\n");
    }
}
/**********************************************************************************
*****************************
 * This function closes USB HMSC on FreeRTOS+FAT and safely ejects.
 **********************************************************************************
*****************************/
```

```
static void usb_safely_eject (void)
{
 /* Check the USB Device Connection before formating */
 if ((true == check_usb_connection()) && (true != b_usb_hmsc_close))
    {
 fsp_err_t freertos_fat_error =
RM_FREERTOS_PLUS_FAT_DiskDeinit(&g_rm_freertos_plus_fat0_ctrl, &my_disk);
 if (FSP_SUCCESS != freertos_fat_error)
     {
           APP_ERR_PRINT("\r\nFREERTOS PLUS FAT DiskDeinit API failed\r\n");
        APP_ERR_TRAP(freertos_fat_error);
     }
 /* Close the FREERTOS_PLUS_FAT_Close instance on safely ejecting */
      freertos_fat_error = RM_FREERTOS_PLUS_FAT_Close
(&g_rm_freertos_plus_fat0_ctrl);
 if (FSP_SUCCESS != freertos_fat_error)
     {
           APP_ERR_PRINT("\r\nFREERTOS PLUS FAT CLOSE API failed\r\n");
        APP_ERR_TRAP(freertos_fat_error);
     }
      APP_PRINT("\r\nUSB Device can be safely removed now\r\n");
 /* Update the flag */
      b_usb_hmsc_close = true;
      g_rm_freertos_plus_fat_insertion_events = false;
   }
 else
   {
 /* USB Device disconnected */
      APP_PRINT("USB Device disconnected or not initialized after Eject command\n");
   }
}
/*******************************************************************************
*******************************
 * This function checks the USB HMSC connection status.
```

```
 ***********************************************************************************************
*****************************/
static bool check_usb_connection (void)
{
 if (true != g_rm_freertos_plus_fat_insertion_events)
    {
 return false;
    }
 else
    {
       APP_PRINT("\r\n USB Device is connected\r\n");
 return true;
    }
}
/***********************************************************************************************
*******************************
 * This function updates write buffer with data and clears read buffer.
 ***********************************************************************************************
*****************************/
static void update_buffer (void)
{
 for (uint16_t i = RESET_VALUE; i < WRITE_ITEM_SIZE; i++)
    {
       g_write_data[i] = ASCII_CHAR_A;
       g_read_data[i] = RESET_VALUE;
    }
}
/***********************************************************************************************
*******************************
 * This function is callback for FreeRTOS+FAT and triggered when USB Pen drive is
removed or inserted.
 ***********************************************************************************************
*****************************/
static void free_rtos_fat_callback (rm_freertos_plus_fat_callback_args_t * p_args)
```

```
{
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED)
    {
       g_rm_freertos_plus_fat_insertion_events = true;
    }
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED)
    {
       g_rm_freertos_plus_fat_insertion_events = false;
    }
}
/**********************************************************************************
*****************************
 * This function Initializes the FreeRTOS+FAT instance..
 **********************************************************************************
*****************************/
static fsp_err_t usb_init (void)
{
 fsp_err_t freertos_fat_error = FSP_SUCCESS;
    int32_t   file_error         = SUCCESS;
 rm_freertos_plus_fat_device_t device;
    memset(&device, RESET_VALUE, sizeof(device));
 /* Open FreeRTOS PLUS FAT */
    freertos_fat_error = RM_FREERTOS_PLUS_FAT_Open(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_cfg);
 if (FSP_SUCCESS != freertos_fat_error)
    {
       APP_ERR_PRINT("\r\nFREERTOS PLUS FAT OPEN API failed\r\n");
 return freertos_fat_error;
    }
    APP_PRINT("\r\n\nFreeRTOS+FAT Open successful\r\n");
 /* Wait for USB Device connection */
    APP_PRINT(" Connect USB Device...\r\n");
 /* Wait until USB Device is connected */
 while (true != check_usb_connection())
```

```
    {
        ;
    }
 /* Initialize the mass storage device. This should not be done until the device is
plugged in and initialized. */
    freertos_fat_error = RM_FREERTOS_PLUS_FAT_MediaInit
(&g_rm_freertos_plus_fat0_ctrl, &device);
 if (FSP_SUCCESS != freertos_fat_error)
    {
        APP_ERR_PRINT("\r\nFreeRTOS Plus FAT Media Init API failed\r\n");
        fat_clean_up();
 return freertos_fat_error;
    }
 /* Initialize one disk for each partition used in the application. */
    freertos_fat_error = RM_FREERTOS_PLUS_FAT_DiskInit(&g_rm_freertos_plus_fat0_ctrl,

&g_rm_freertos_plus_fat0_disk_cfg,
                                                       &my_disk);
 if (FSP_SUCCESS != freertos_fat_error)
    {
        APP_ERR_PRINT("\r\nFreeRTOS Plus FAT Disk Init API failed\r\n");
        fat_clean_up();
 return freertos_fat_error;
    }
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&my_disk, my_disk.xStatus.bPartitionNumber);
 if (FSP_SUCCESS != ff_err)
    {
        APP_ERR_PRINT("\r\nFF_Mount API failed\r\n");
 /* Close the FREERTOS_PLUS_FAT_Close instance on safely ejecting */
        fat_clean_up();
 /* As function returns fsp_err_t, ff_err cannot be returned. Hence trapping the
error here */
        APP_ERR_TRAP(ff_err);
```

```
    }
 /* Add the disk to the file system. */
    file_error = FF_FS_Add("/", &my_disk);
 if (SUCCESS == file_error)
    {
       APP_ERR_PRINT("\r\nFF_Mount API failed\r\n");
 /* Close the FREERTOS_PLUS_FAT_Close instance on safely ejecting */
       fat_clean_up();
 /* As function returns fsp_err_t, ff_err cannot be returned. Hence trapping the
error here */
       APP_ERR_TRAP(file_error);
    }
 /* Set this flag to let application know that USB is initialized */
    b_usb_hmsc_close = false;
 return freertos_fat_error;
}
/**********************************************************************************
*****************************
 * This function closes the FreeRTOS+FAT instance..
 **********************************************************************************
*****************************/
static void fat_clean_up (void)
{
 fsp_err_t freertos_fat_error = FSP_SUCCESS;
 /* Close the FREERTOS_PLUS_FAT_Close instance on any failure */
    freertos_fat_error = RM_FREERTOS_PLUS_FAT_Close(&g_rm_freertos_plus_fat0_ctrl);
 if (FSP_SUCCESS != freertos_fat_error)
    {
       APP_PRINT("\r\nFREERTOS PLUS FAT CLOSE API failed\r\n");
    }
 else
    {
       APP_PRINT("\r\nFREERTOS PLUS FAT instance Closed successfully.\r\n");
    }
```

```
}
/***********************************************************************************
 * Event notification thread for Interrupt IN/OUT test.
 **********************************************************************************/
static void usb_app_hcdc_task (void * pvParameters)
{
 fsp_err_t                err       = FSP_SUCCESS;
 static usb_event_info_t * event_info = NULL;
    BaseType_t                err_queue = pdFALSE;
    memset(&g_serial_state, RESET_VALUE, sizeof(g_serial_state));
    memset(&g_com_parm, RESET_VALUE, sizeof(g_com_parm));
    g_snd_buf[ZERO_INDEX] = KIT_INFO;
    uint32_t i;
 /*Fill the write buffer*/
 for (i = RESET_VALUE; i < CDC_WRITE_DATA_LEN; i++)
    {
       g_snd_buf[i] = (uint8_t) i;
    }
 while (true)
    {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
     {
           handle_error(g_err_flag, "Error in sending usb event through queue");
     }
 /* Receive message from queue and analyzing the received message*/
      err_queue = xQueueReceive(g_usb_queue, &event_info, (portMAX_DELAY));
 /* Handle error */
 if (pdTRUE != err_queue)
     {
           handle_error(err_queue, "Error in receiving USB event message through
queue");
     }
 switch (event_info->event)
```

```
     {
 case USB_STATUS_CONFIGURED:
     {
                vTaskDelay(200);       // GR_debug
 /* CDC Class request "SetLineCoding" */
                set_line_coding(&g_basic0_ctrl, event_info->device_address);
 break;
     }
 case USB_STATUS_READ_COMPLETE:
     {
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
     {
 /* CDC class communication data process */
                  usb_data_process(event_info);
     }
 break;
     }
 case USB_STATUS_WRITE_COMPLETE:
     {
 if (FSP_SUCCESS == event_info.status)
     {
 /* Report receive start */
                  err = R_USB_Read(&g_basic0_ctrl, g_rcv_buf, CDC_READ_DATA_LEN,
event_info->device_address);
 /* Handle Error */
 if (FSP_SUCCESS != err)
     {
                      handle_error(err, "**R_USB_Read API FAILED**");
     }
     }
 break;
     }
 case USB_STATUS_REQUEST_COMPLETE:
```

```
     {
 /* Check Complete request "SetLineCoding" */
 if (USB_CDC_SET_LINE_CODING == (event_info->setup.request_type & USB_BREQUEST))
     {
 /* Class notification "SerialState" receive start */
                  set_control_line_state(&g_basic0_ctrl,
event_info->device_address);
     }
 /* Check Complete request "SetControlLineState" */
 else if (USB_CDC_SET_CONTROL_LINE_STATE == (event_info->setup.request_type &
USB_BREQUEST))
     {
 /* CDC Class request "SetLineCoding" */
                  get_line_coding(&g_basic0_ctrl, event_info->device_address);
     }
 else if (USB_CDC_GET_LINE_CODING == (event_info->setup.request_type & USB_BREQUEST))
     {
                  err = R_USB_Write(&g_basic0_ctrl, g_snd_buf, CDC_WRITE_DATA_LEN,
event_info->device_address);
 if (FSP_SUCCESS != err)
     {
                      handle_error(err, "**R_USB_Write API FAILED**");
     }
     }
 else
     {
 /* Not support request */
     }
 break;
     }
 default:
     {
 /* No operation to do*/
 break;
```

```
      }
      }
       vTaskDelay(200);
    }
}
/*********************************************************************************
*****************************
 * In this function initializes to set control line state information for host
control transfer.
 *********************************************************************************
*****************************/
static void set_control_line_state (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
 fsp_err_t   err = FSP_SUCCESS;
    setup.request_type   = SET_CONTROL_LINE_STATE; /*
bRequestCode:SET_CONTROL_LINE_STATE, bmRequestType */
    setup.request_value = VALUE_ZERO;             /* wValue:Zero */
    setup.request_index = VALUE_ZERO;             /* wIndex:Interface */
    setup.request_length = VALUE_ZERO;             /* wLength:Zero */
    err = R_USB_HostControlTransfer(p_ctrl, &setup, &g_usb_dummy, device_address);
 if (FSP_SUCCESS != err)
    {
       handle_error(err, "**R_USB_HOSTControlTransfer API FAILED**");
    }
}
/*********************************************************************************
*****************************
 * In this function initializes to set line coding information for host control
transfer.
 *********************************************************************************
*****************************/
static void set_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
```

```
{
    usb_setup_t setup;
 fsp_err_t   err = FSP_SUCCESS;
    g_com_parm.dwdte_rate   = (uint32_t) USB_HCDC_SPEED_9600;
    g_com_parm.bdata_bits   = USB_HCDC_DATA_BIT_8;
    g_com_parm.bchar_format = USB_HCDC_STOP_BIT_1;
    g_com_parm.bparity_type = USB_HCDC_PARITY_BIT_NONE;
    setup.request_type   = SET_LINE_CODING;    /* bRequestCode:SET_LINE_CODING,
bmRequestType */
    setup.request_value = VALUE_ZERO;         /* wValue:Zero */
    setup.request_index = VALUE_ZERO;         /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    err = R_USB_HostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
 if (FSP_SUCCESS != err)
    {
       handle_error(err, "**R_USB_HostControlTransfer API FAILED**");
    }
} /* End of function cdc_set_line_coding */
/***********************************************************************************
*******************************
 * In this function initializes to get line coding information for host control
transfer.
 ***********************************************************************************
*****************************/
static void get_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
 fsp_err_t   err = FSP_SUCCESS;
    setup.request_type   = GET_LINE_CODING;    /* bRequestCode:GET_LINE_CODING,
bmRequestType */
    setup.request_value = VALUE_ZERO;         /* wValue:Zero */
    setup.request_index = VALUE_ZERO;         /* wIndex:Interface */
```

```
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    err = R_USB_HostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
 if (FSP_SUCCESS != err)
    {
       handle_error(err, "**R_USB_HostControlTransfer API FAILED**");
    }
}
/**********************************************************************************
*****************************
 * This function is called to do closing of usb module using its HAL level API and
handles the error trap.
 * Handle the Error internally with Proper Message. Application handles the rest.
 **********************************************************************************
*****************************/
static void handle_error (fsp_err_t err, char * err_str)
{
}
/**********************************************************************************
*****************************
 * This function is to do data process with peripheral device
 **********************************************************************************
*****************************/
static void usb_data_process (usb_event_info_t * event_info)
{
 fsp_err_t err = FSP_SUCCESS;
 if (USB_CLASS_HCDC == event_info->type)
    {
 if (RESET_VALUE < event_info->data_size)
      {
 if (0 != memcmp(g_rcv_buf, g_snd_buf, event_info->data_size))
      {
                APP_PRINT("\r\n Sending and receiving data do not match :%s",
```

```
g_rcv_buf);
     }
           err = R_USB_Write(&g_basic0_ctrl, g_snd_buf, CDC_WRITE_DATA_LEN,
event_info->device_address);
 if (FSP_SUCCESS != err)
     {
          handle_error(err, "**R_USB_Write API FAILED**");
     }
           APP_PRINT("\r\n Received data :%s", g_rcv_buf);
     }
 else
     {
 /* Send the data reception request when the zero-length packet is received. */
           err = R_USB_Read(&g_basic0_ctrl, g_rcv_buf, event_info->data_size,
event_info->device_address);
 if (FSP_SUCCESS != err)
     {
          handle_error(err, "**R_USB_Read API FAILED**");
     }
     }
   }
 else
   {
 /* Class notification "SerialState" receive start */
      err = R_USB_Read(&g_basic0_ctrl,
                       (uint8_t *) &g_serial_state,
                       USB_HCDC_SERIAL_STATE_MSG_LEN,
                       event_info->device_address);
 /* Error Handle */
 if (FSP_SUCCESS != err)
     {
           handle_error(err, "**R_USB_Read API FAILED**");
     }
   }
```

```
} /* End of function usb_data_process */
/******************************************************************************
*****************************
 * This function is callback for FreeRTOS+HCDC and triggers when USB event occurs
from the device.
 ******************************************************************************
*****************************/
static void usb_app_common_callback (usb_event_info_t * p_event_info, usb_hdl_t
cur_task, usb_onoff_t usb_state)
{
 FSP_PARAMETER_NOT_USED(cur_task);
 FSP_PARAMETER_NOT_USED(usb_state);
 if (USB_CLASS_REQUEST == p_event_info->type)
    {
 if ((USB_CDC_SET_LINE_CODING == (p_event_info->setup.request_type & USB_BREQUEST))
||
           (USB_CDC_GET_LINE_CODING == (p_event_info->setup.request_type &
USB_BREQUEST)) ||
           (USB_CDC_SET_CONTROL_LINE_STATE == (p_event_info->setup.request_type &
USB_BREQUEST)))
     {
           g_usb_host_apl_callback[0](p_event_info, cur_task, usb_state); // HCDC
Callback
     }
 else
     {
           g_usb_host_apl_callback[1](p_event_info, cur_task, usb_state); // HMSC
Callback
     }
    }
 else
    {
 if ((USB_CLASS_HCDC == p_event_info->type) || (USB_CLASS_HCDCC ==
p_event_info->type))
```

```
      {
            g_usb_host_apl_callback[0](p_event_info, cur_task, usb_state); // HCDC
Callback
      }
 else
      {
            g_usb_host_apl_callback[1](p_event_info, cur_task, usb_state); // HMSC
Callback
      }
    }
}
/**********************************************************************************
*****************************
 * This function is callback for FreeRTOS+HCDC and triggers when USB event occurs
from the device.
 **********************************************************************************
*****************************/
static void usb_app_hcdc_callback (usb_event_info_t * p_event_info, usb_hdl_t
cur_task, usb_onoff_t usb_state)
{
 FSP_PARAMETER_NOT_USED(cur_task);
 FSP_PARAMETER_NOT_USED(usb_state);
 /* Send event received to queue */
 if (pdTRUE != (xQueueSend(g_usb_queue, (const void *) &p_event_info, (TickType_t)
(NO_WAIT_TIME))))
    {
       g_err_flag = true;
    }
}
```

### Typedefs

| | |
|---|---|
| typedef usb_event_info_t | usb_instance_ctrl_t |

### Typedef Documentation

### ◆ usb_instance_ctrl_t

| typedef usb_event_info_t usb_instance_ctrl_t |
|---|
| USB private control block. DO NOT MODIFY. Initialization occurs when R_USB_Open is called. |

## Function Documentation

### ◆ R_USB_EventGet()

| fsp_err_t R_USB_EventGet ( usb_ctrl_t *const *p_api_ctrl*, usb_status_t * *event* ) |
|---|
| Obtains completed USB related events. (OS-less Only)<br><br>In USB host mode, the device address value of the USB device that completed an event is specified in the usb_ctrl_t structure member (address) specified by the event's argument. In USB peripheral mode, USB_NULL is specified in member (address). If this function is called in the RTOS execution environment, a failure is returned.<br><br>**Return values**<br><br>FSP_SUCCESS — Event Get Success.<br>FSP_ERR_USB_FAILED — If called in the RTOS environment, an error is returned.<br><br>***Note***<br>*Do not use the same variable as the first argument of R_USB_Open for the first argument.*<br>*Do not call this API in the interrupt function.* |

### ◆ R_USB_Callback()

| fsp_err_t R_USB_Callback ( usb_callback_t * *p_callback*) |
|---|
| Register a callback function to be called upon completion of a USB related event. (RTOS only)<br><br>This function registers a callback function to be called when a USB-related event has completed. If this function is called in the OS-less execution environment, a failure is returned.<br><br>**Return values**<br><br>FSP_SUCCESS — Successfully completed.<br>FSP_ERR_ASSERTION — Parameter is NULL error.<br><br>***Note***<br>*Do not call this API in the interrupt function.* |

### ◆ R_USB_Open()

fsp_err_t R_USB_Open ( usb_ctrl_t *const *p_api_ctrl*, usb_cfg_t const *const *p_cfg* )

Applies power to the USB module specified in the argument (p_ctrl).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success in open. |
| FSP_ERR_USB_BUSY | Specified USB module now in use. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_Close()

fsp_err_t R_USB_Close ( usb_ctrl_t *const *p_api_ctrl*)

Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_NOT_OPEN | USB module is not open. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_Read()

fsp_err_t R_USB_Read ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *destination* )

Bulk/Interrupt data transfer.

Requests USB data read (bulk/interrupt transfer). The read data is stored in the area specified by argument (p_buf). After data read is completed, confirm the operation by checking the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function. The received data size is set in member (size) of the usb_ctrl_t structure. To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed (Data read request completed). |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Data receive request already in process for USB device with same device address. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*1. Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*
*2. Allocate the following the storage area when using DMA transfer and specify the start address of the allocated storage area to the 2nd argument(p_buf).*
*(1). When using High-speed and enabling continuous transfer mode, allocate the storage area with a size that is a multiple of 2048.*
*(2). When using High-speed and disabling continuous transfer mode, allocate the storage area with a size that is a multiple of 512.*
*(3). When using Full-speed, allocate the storage area with a size that is a multiple of 64.*
*3. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.*
*(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.*
*(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.*

### ◆ R_USB_Write()

fsp_err_t R_USB_Write ( usb_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_buf*, uint32_t *size*, uint8_t *destination* )

Bulk/Interrupt data transfer.

Requests USB data write (bulk/interrupt transfer). Stores write data in area specified by argument (p_buf). Set the device class type in usb_ctrl_t structure member (type). Confirm after data write is completed by checking the return value (USB_STATUS_WRITE_COMPLETE) of the R_USB_GetEvent function. For sending a zero-length packet, please refer the following Note.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (Data write request completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Data write request already in process for USB device with same device address. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*1. The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_Write function as follow. e.g)*
*R_USB_Write (&g_basic0_ctrl, &g_buf, USB_NULL);*
*2. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.*
*(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.*
*(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.*
*3. Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_Stop()

fsp_err_t R_USB_Stop ( usb_ctrl_t *const *p_api_ctrl*, usb_transfer_t *direction*, uint8_t *destination* )

Requests a data read/write transfer be terminated when a data read/write transfer is being performed.

To stop a data read, set USB_TRANSFER_READ as the argument (type); to stop a data write, specify USB_WRITE as the argument (type).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (stop completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Stop processing is called multiple times. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_Suspend()

fsp_err_t R_USB_Suspend ( usb_ctrl_t *const *p_api_ctrl*)

Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure.

After the suspend request is completed, confirm the operation with the return value (USB_STATUS_SUSPEND) of the R_USB_EventGet function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | During a suspend request to the specified USB module, or when the USB module is already in the suspended state. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_Resume()

fsp_err_t R_USB_Resume ( usb_ctrl_t *const *p_api_ctrl*)

Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure.

After the resume request is completed, confirm the operation with the return value (USB_STATUS_RESUME) of the R_USB_EventGet function

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Resume already requested for same device address. (USB host mode only) |
| FSP_ERR_USB_NOT_SUSPEND | USB device is not in the SUSPEND state. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_VbusSet()

fsp_err_t R_USB_VbusSet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t *state* )

Specifies starting or stopping the VBUS supply.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (VBUS supply start/stop completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_InfoGet()

fsp_err_t R_USB_InfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_info_t * *p_info*, uint8_t *destination* )

Obtains completed USB-related events.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (VBUS supply start/stop completed) |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PipeRead()

fsp_err_t R_USB_PipeRead ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number* )

Requests a data read (Bulk/Interrupt transfer) via the pipe specified in the argument.

The read data is stored in the area specified in the argument (p_buf). After the data read is completed, confirm the operation with the R_USB_GetEvent function return value(USB_STATUS_READ_COMPLETE). To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*1. Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*
*2. Allocate the following the storage area when using DMA transfer and specify the start address of the allocated storage area to the 2nd argument(p_buf).*
*(1). When using High-speed and enabling continuous transfer mode, allocate the storage area with a size that is a multiple of 2048.*
*(2). When using High-speed and disabling continuous transfer mode, allocate the storage area with a size that is a multiple of 512.*
*(3). When using Full-speed, allocate the storage area with a size that is a multiple of 64.*
*3. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.*
*(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.*
*(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.*

### ◆ R_USB_PipeWrite()

fsp_err_t R_USB_PipeWrite ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number* )

Requests a data write (Bulk/Interrupt transfer).

The write data is stored in the area specified in the argument (p_buf). After data write is completed, confirm the operation with the return value (USB_STATUS_WRITE_COMPLETE) of the EventGet function. For sending a zero-length packet, please refer the following Note.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*1. The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_PipeWrite function as follow.*
*e.g)*
*R_USB_PipeWrite (&g_basic0_ctrl, &g_buf, USB_NULL, pipe_number);*
*2. Specify the following address to the 2nd argument (p_buf) when using DMA transfer.*
*(1). When using High-speed module, specify start address of the buffer area aligned on 4-byte boundary.*
*(2). When using Full-speed module, specify start address of the buffer area aligned on 2-byte boundary.*
*3. Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_PipeStop()

fsp_err_t R_USB_PipeStop ( usb_ctrl_t *const *p_api_ctrl*, uint8_t *pipe_number* )

Terminates a data read/write operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. (Stop request completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_UsedPipesGet()

fsp_err_t R_USB_UsedPipesGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_pipe*, uint8_t *destination* )

Gets the selected pipe number (number of the pipe that has completed initialization) via bit map information.

The bit map information is stored in the area specified in argument (p_pipe). Based on the information (module member and address member) assigned to the usb_ctrl_t structure, obtains the PIPE information of that USB device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PipeInfoGet()

fsp_err_t R_USB_PipeInfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_pipe_t * *p_info*, uint8_t *pipe_number* )

Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size.

The obtained pipe information is stored in the area specified in the argument (p_info).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_PullUp()

fsp_err_t R_USB_PullUp ( usb_ctrl_t *const *p_api_ctrl*, uint8_t *state* )

This API enables or disables pull-up of D+/D- line.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. (Pull-up enable/disable setting completed) |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_HostControlTransfer()

fsp_err_t R_USB_HostControlTransfer ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *p_setup*, uint8_t * *p_buf*, uint8_t *device_address* )

Performs settings and transmission processing when transmitting a setup packet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_PeriControlDataGet()

fsp_err_t R_USB_PeriControlDataGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size* )

Receives data sent by control transfer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_PeriControlDataSet()

fsp_err_t R_USB_PeriControlDataSet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size* )

Performs transfer processing for control transfer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_BUSY | Specified pipe now handling data receive/send request. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_PeriControlStatusSet()

fsp_err_t R_USB_PeriControlStatusSet ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_status_t *status* )

Set the response to the setup packet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_RemoteWakeup()

| fsp_err_t R_USB_RemoteWakeup ( usb_ctrl_t *const *p_api_ctrl*) |
|---|

Sends a remote wake-up signal to the connected Host.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_NOT_SUSPEND | Device is not suspended. |
| FSP_ERR_USB_BUSY | The device is in resume operation. |

***Note***

*Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_DriverActivate()

| fsp_err_t R_USB_DriverActivate ( usb_ctrl_t *const *p_api_ctrl*) |
|---|

Activate USB Driver for USB Peripheral BareMetal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |

***Note***

*Call this API in the in the infinite loop of the application program or a timer interrupt.*

### ◆ R_USB_CallbackMemorySet()

fsp_err_t R_USB_CallbackMemorySet ( usb_ctrl_t *const *p_api_ctrl*, usb_callback_args_t * *p_callback_memory* )

Set callback memory to USB Driver for USB Peripheral BareMetal.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |

***Note***

*Call this API after calling R_USB_Open function.*

### ◆ R_USB_ModuleNumberGet()

fsp_err_t R_USB_ModuleNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *module_number* )

This API gets the module number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

### ◆ R_USB_ClassTypeGet()

fsp_err_t R_USB_ClassTypeGet ( usb_ctrl_t *const *p_api_ctrl*, usb_class_t * *class_type* )

This API gets the class type.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_DeviceAddressGet()

fsp_err_t R_USB_DeviceAddressGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *device_address* )

This API gets the device address.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_PipeNumberGet()

fsp_err_t R_USB_PipeNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *pipe_number* )

This API gets the pipe number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_DeviceStateGet()

fsp_err_t R_USB_DeviceStateGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *state* )

This API gets the state of the device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_DataSizeGet()

fsp_err_t R_USB_DataSizeGet ( usb_ctrl_t *const *p_api_ctrl*, uint32_t * *data_size* )

This API gets the read data size.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_SetupGet()

fsp_err_t R_USB_SetupGet ( usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *setup* )

This API gets the setup information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |

***Note***

*In Bare-Metal, In the Bare-Metal version, please specify the variable specified by the 1st argument of the R_USB_EventGet function to the 1st argument of this API.*
*In the FreeRTOS, please specify one of the following to the 1st argument of this API.*
*1. The 1st argument of the callback function specified in Conguration.*
*2. The start address of the area where the structure area of the 1st argument was copied.*

### ◆ R_USB_OtgCallbackSet()

fsp_err_t R_USB_OtgCallbackSet ( usb_ctrl_t *const *p_api_ctrl*, usb_otg_callback_t * *p_callback* )

Set callback function to be called when the OTG role swap was completed on Azure RTOS.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |

### ◆ R_USB_OtgSRP()

| fsp_err_t R_USB_OtgSRP ( usb_ctrl_t *const *p_api_ctrl*) |
|---|
| Start the SRP processing for OTG on Azure RTOS.<br><br>**Return values**<br>FSP_SUCCESS — Successful completion.<br>FSP_ERR_USB_FAILED — The function could not be completed successfully.<br>FSP_ERR_ASSERTION — Parameter is NULL error.<br><br>***Note***<br>*Do not support the VBUS Pulsing since OTG 2.0 does not support the VBUS Pulsing..* |

#### 5.2.6.34 USB Composite (r_usb_composite)

Modules » Connectivity

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

# Overview

USB composite device works as a USB Peripheral by combining two peripheral device classes and r_usb_basic module.
This USB driver supports the following composite devices:

1. PCDC + PMSC
2. PCDC + PHID
3. PHID + PMSC
4. PCDC + PCDC
5. PHID + PHID
6. PCDC + PVND

# How to Configuration

The following shows FSP configuration procedure for USB composite device.

- Select [New Stack]->[USB]->[USB Composite]

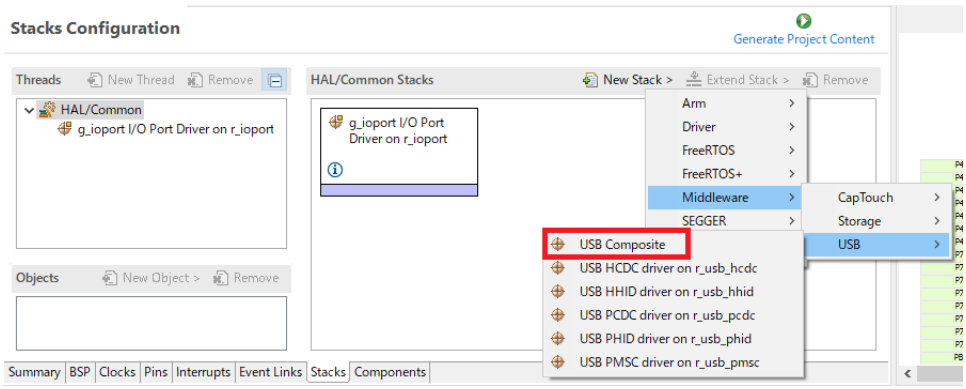



Figure 230: Select USB Composite

- The following is displayed when selecting [USB Composite].

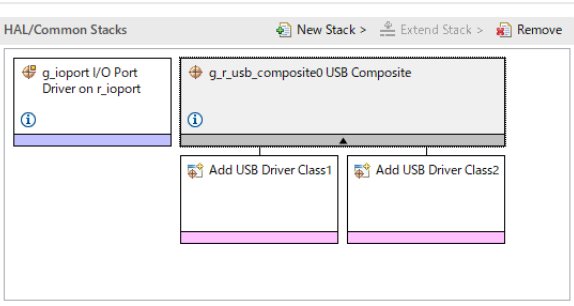



Figure 231: USB Composite Stack

- Select the supported 2 device classes as follows.

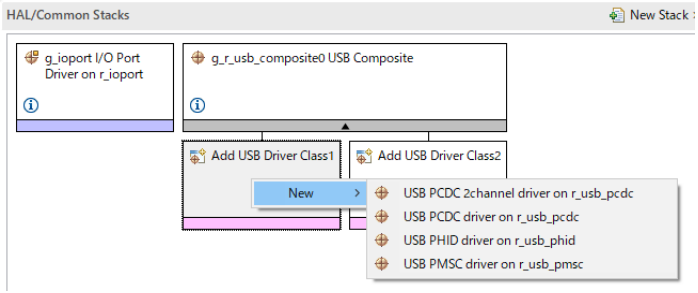



Figure 232: Select Device Classes

*Note*

1. *Be sure to select "USB PCDC driver on r_usb_pcdc" and "USB PCDC 2channel driver on r_usb_pcdc" when configurating for "PCDC + PCDC".*
2. *Be sure to select "USB PHID driver on r_usb_phid" and "USB PHID 2channel driver on r_usb_phid" when configurating for "PHID + PHID".*

- Select the supported 2 device classes as follows. The following is displayed when selecting

2 device classes.

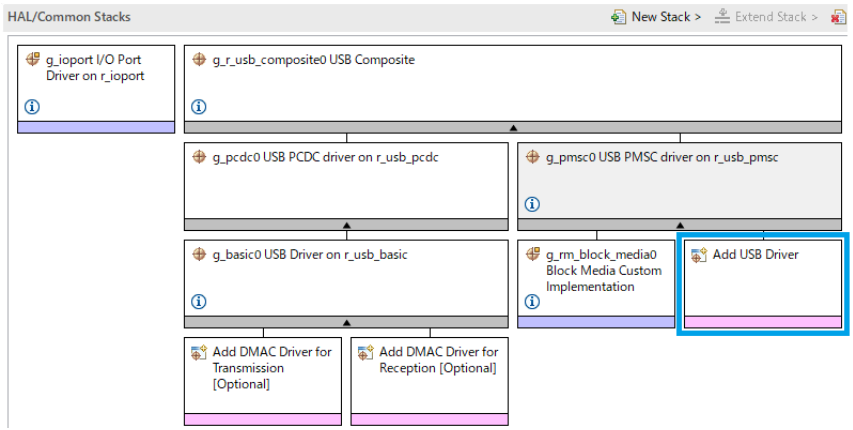



Figure 233: Delete USB Basic Instance

*Note*

1. *Delete the "g_basic1" instance manually since this instance is not used in composite device. (Refer to the blue frame in the above figure.)*
2. *The error is output when selecting the following device classes.*
    a. *PMSC + PMSC*

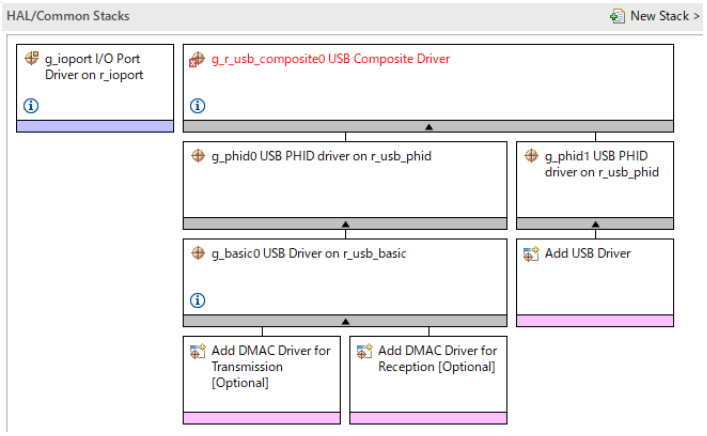



Figure 234: Device Class Selection Error

## Limitations

- The following composite device is not suppored when using RA2A1(MCU).
    - PMSC + PCDC
    - PCDC + PCDC

- If you use PMSC, make sure to use usb_basic module with PMSC.

  There is a risk that the information on the PMSC storage media cannot be registered normally in the "USB Callback Context".

**Notes**

Please determine by refer to the member "type" in "usb_event_info" structure when getting PCDC channel number which the write event is completed in PCDC + PCDC, PHID + PHID, etc.

# Descriptor

Templates for composite device descriptors can be found in ra/fsp/src/r_usb_composite folder. Also, please be sure to use your vendor ID.

1. r_usb_pcdc_pmsc_descriptor.c.template (for PCDC + PMSC)
2. r_usb_pcdc_phid_descriptor.c.template (for PCDC + PHID)
3. r_usb_phid_pmsc_descriptor.c.template (for PHID + PMSC)
4. r_usb_pcdc_pcdc_descriptor.c.template (for PCDC + PCDC)
5. r_usb_phid_phid_descriptor.c.template (for PHID + PHID)
6. r_usb_pcdc_pvnd_descriptor.c.template (for PCDC + PVND) and r_usb_pcdc_pvnd_descriptor.h.template (for PCDC + PVND)

# Examples

### USB COMPOSITE Example

- PCDC + PHID

```
void main_task (void)
{
  #if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
  #endif
    usb_event_info_t usb_event;
 usb_status_t    event;
    uint8_t       * p_idle_value;
    uint8_t          send_data[16] BSP_ALIGN_VARIABLE(4);
    uint8_t          req_comp_flag = 0;
    uint8_t          count         = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf_phid);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
  #if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
```

```
      usb_event = *p_mess;
event    = usb_event.event;
 #else  /* (BSP_CFG_RTOS == 2) */
R_USB_EventGet(&usb_event, &event);
 #endif /* (BSP_CFG_RTOS == 2) */
switch (event)
     {
case USB_STATUS_CONFIGURED:
     {
               g_status = NO_WRITING;
               g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
break;
           }
case USB_STATUS_WRITE_COMPLETE:
     {
if (FSP_SUCCESS == usb_event.status)
     {
if (usb_event.type == USB_CLASS_PCDC)
     {
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
             }
else if (usb_event.type == USB_CLASS_PHID)
     {
if (DATA_WRITING == g_status)
     {
                          g_status = ZERO_WRITING;
                          g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *)
g_zero_data, DATA_LEN_PHID, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
             }
else if (g_status == ZERO_WRITING)
     {
                g_status = NO_WRITING;
             }
             }
```

```
            }
 break;
           }
 case USB_STATUS_READ_COMPLETE:
     {
 if ((FSP_SUCCESS == usb_event.status) || (FSP_ERR_USB_SIZE_SHORT ==
usb_event.status))
     {
 if (usb_event.type == USB_CLASS_PCDC)
     {
                      g_usb_on_usb.write(&g_basic0_ctrl, g_buf,
usb_event.data_size, USB_CLASS_PCDC);
 if (req_comp_flag == 1)
     {
 if (g_status == NO_WRITING)
     {
                           count++;
                           g_status = DATA_WRITING;
                           g_usb_on_usb.write(&g_basic0_ctrl, g_buf_phid,
DATA_LEN_PHID, USB_CLASS_PHID);
            }
            }
            }
            }
 break;
           }
 case USB_STATUS_REQUEST:   /* Receive Class Request */
     {
 if (USB_PCDC_SET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
     {
 R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
            }
 else if (USB_PCDC_GET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
```

```
     {
 R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
             }
 else if (USB_SET_REPORT == (usb_event.setup.request_type & USB_BREQUEST))
     {
                 g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
             }
 else if (USB_GET_DESCRIPTOR == (usb_event.setup.request_type & USB_BREQUEST))
     {
 if (USB_GET_REPORT_DESCRIPTOR == usb_event.setup.request_value)
     {
                     g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                     (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
             }
 else if (USB_GET_HID_DESCRIPTOR == usb_event.setup.request_value)
     {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
     {
                         send_data[i] = g_apl_configuration[84 + i];
             }
 /* Configuration Descriptor address set. */
                     g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
             }
 else
     {
                     g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
             }
             }
```

```
 else if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
 /* Get SetIdle value */
                  p_idle_value = (uint8_t *) &usb_event.setup.request_value;
                  g_idle       = p_idle_value[1];
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
              }
 else if (USB_GET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
              }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
              }
 else if (USB_GET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
              }
 else
      {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
              }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
      {
 if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
                  p_idle_value = (uint8_t *) &usb_event.setup.request_value;
```

```
                    g_idle = p_idle_value[1];
                }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
 /* None */
 /* g_protocol = event_info.setup.value; */
                }
 else
      {
                    req_comp_flag = 1;
                }
 break;
            }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
      {
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main() */
void set_key_data (uint8_t * p_buf)
{
 static uint8_t key_data;
    key_data = KBD_CODE_A;
    *(p_buf + 2) = key_data;
}
  #if (BSP_CFG_RTOS == 2)
/***********************************************************************************
 * Function Name : usb_apl_rec_msg
```

```
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ***********************************************************************/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
    (void) tm;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (portMAX_DELAY));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/***********************************************************************
 * End of function usb_apl_rec_msg
 ***********************************************************************/
/***********************************************************************
 * Function Name : usb_apl_snd_msg
```

```
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
 ***********************************************************************/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/***********************************************************************
 * End of function usb_apl_snd_msg
 ***********************************************************************/
  #endif  /* #if (BSP_CFG_RTOS == 2) */
```

- PCDC + PVND

```
/***********************************************************************
```

```
 * Includes <System Includes> , "Project Includes"
 ***********************************************************************************/
#include "r_usb_basic.h"
#include "r_usb_pcdc_api.h"
#include "r_usb_pcdc_cfg.h"
#include "r_usb_pcdc_pvnd_descriptor.h"
/***********************************************************************************
 * Macro definitions
 ***********************************************************************************/
#define DATA_LEN 2048
#define RESET_VALUE 0
#define INT_EVT_TSK_STACK_SIZE 1024  /* Stack size of Event notification thread for
vendor class's Interrupt IN/OUT test */
#define INT_EVT_TSK_PRI 2  /* Priority of Event notification thread for vendor
class's Interrupt IN/OUT test */
#define BUF_SIZE (2048)  /* Buffer size */
#define REQ_SIZE (20)  /* Request buffer size */
#define USB_VALUE_FF (0xFFU)  /* FF macro */
#define USB_APL_MXPS (64U)
#define START_PIPE (USB_PIPE1)  /* Start pipe number */
#define END_PIPE (USB_PIPE9 + 1) /* Total pipe */
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#define SET_VENDOR_NO_DATA (USB_SET_VENDOR_NO_DATA | USB_HOST_TO_DEV | USB_VENDOR |
USB_INTERFACE)
#define SET_VENDOR (USB_SET_VENDOR | USB_HOST_TO_DEV | USB_VENDOR | USB_INTERFACE)
#define GET_VENDOR (USB_GET_VENDOR | USB_DEV_TO_HOST | USB_VENDOR | USB_INTERFACE)
#define DELAY (10U) /* Delay for print */
#define USB_STATUS_VENDOR_INTIN_TEST ((usb_status_t) 0x01)
#define USB_STATUS_VENDOR_INTOUT_TEST ((usb_status_t) 0x02)
#define USB_STATUS_COM_IN_TEST ((usb_status_t) 0x03)
#define USB_STATUS_RE_INIT_TEST ((usb_status_t) 0x04)
```

```
#define HS_MAX_PACKET_SIZE (512)
#define FS_MAX_PACKET_SIZE (64)
#define VALUE_A1H (0xA1)
#define LINE_CODING_LENGTH (0x07U)
/*******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
static uint8_t g_cdc_buf[BUF_SIZE] = {RESET_VALUE}; /* Data buffer for PCDC Normal
Request */
static uint8_t g_vnd_buf[BUF_SIZE] = {RESET_VALUE}; /* Data buffer for PVND Normal
Request */
static uint8_t g_request_buf[REQ_SIZE] = {RESET_VALUE}; /* Data buffer for PVND Class
Request */
static uint8_t g_apl_vendor_bulk_in = RESET_VALUE; /* Bulk In Pipe */
static uint8_t g_apl_vendor_bulk_out = RESET_VALUE; /* Bulk Out Pipe */
static uint8_t g_apl_vendor_interrupt_in = RESET_VALUE; /* Interrupt In Pipe */
static uint8_t g_apl_vendor_interrupt_out = RESET_VALUE; /* Interrupt Out Pipe */
static uint8_t g_apl_cdc_interrupt_in = RESET_VALUE; /* CDC Interrupt In Pipe */
static uint16_t g_max_packet_size = USB_APL_MXPS;
/* Variable to capture USB event. */
static volatile usb_event_info_t * p_usb_event = NULL;
static volatile bool g_err_flag = false;
static bool b_usb_attach = false;
static usb_pcdc_linecoding_t g_line_coding;
#if (BSP_CFG_RTOS == 2)
static TaskHandle_t g_app_interrupt_event_notify_tsk_hdl;
#endif  /* (BSP_CFG_RTOS == 2) */
static uint32_t g_interrupt_in_test_flag = 0;
static uint32_t g_interrupt_out_test_flag = 0;
static uint32_t g_interrupt_out_times = 0;
static uint32_t g_interrupt_com_event = 0;
static uint8_t g_interrupt_in_com_data[8] = {VALUE_A1H, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00};
static uint32_t g_test_reinit_flag = 0;
```

```
/* Interrupt Event for Application */
#if (BSP_CFG_RTOS == 2)
static usb_event_info_t g_event_info2;
static usb_event_info_t * g_p_event_info2;
#else
static usb_event_info_t g_usb_event;
static usb_status_t g_event;
static uint32_t g_event2;
#endif
extern const usb_cfg_t g_basic0_cfg;
usb_instance_ctrl_t g_basic0_ctrl;
/* Function definitions */
static fsp_err_t process_usb_events(void);
static fsp_err_t process_usb_events_for_interrupt(void);
static fsp_err_t usb_configured_event_process(void);
static fsp_err_t usb_status_request(void);
static void handle_error(fsp_err_t err);
static fsp_err_t buffer_check(uint32_t length);
void usb_composite_thread_entry(void * pvParameters);
void usb_app_interrupt_event_task(void * pvParameters);
#if (BSP_CFG_RTOS == 2)
void usb_composite_callback(usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t on_off);
#endif  /* (BSP_CFG_RTOS == 2) */
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_app_interrupt_event_task
 * Description : Event notification thread for Interrupt IN/OUT test.
 * Argument : void * pvParameters : Ipointer to pvParameters
 * Return : none
 ******************************************************************************/
void usb_app_interrupt_event_task (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
```

```
 while (1)
    {
 if (1 == g_interrupt_in_test_flag)
     {
           g_interrupt_in_test_flag = 0;
           g_event_info2.event = USB_STATUS_VENDOR_INTIN_TEST;
           g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-IN event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
     {
              g_err_flag = true;
           }
      }
 if (1 == g_interrupt_out_test_flag)
     {
           g_interrupt_out_test_flag = 0;
           g_event_info2.event = USB_STATUS_VENDOR_INTOUT_TEST;
           g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-OUT event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
     {
              g_err_flag = true;
           }
      }
 if (0 != g_interrupt_com_event)
     {
           g_interrupt_com_event = 0;
           g_event_info2.event = USB_STATUS_COM_IN_TEST;
           g_p_event_info2    = &g_event_info2;
 /* Send Interrupt-IN event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
```

```
     {
               g_err_flag = true;
           }
      }
 if (0 != g_test_reinit_flag)
     {
           g_test_reinit_flag = 0;
           g_event_info2.event = USB_STATUS_RE_INIT_TEST;
           g_p_event_info2    = &g_event_info2;
 /* Send RE_INIT_TEST event to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &g_p_event_info2,
(TickType_t) (RESET_VALUE))))
     {
               g_err_flag = true;
           }
      }
       vTaskDelay(100);
    }
}
/******************************************************************************
 * End of function usb_app_interrupt_event_task
 ******************************************************************************/
#endif  /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_composite_thread_entry
 * Description : Peripheral CDC & Vendor application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_composite_thread_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 fsp_err_t err = FSP_SUCCESS;
#if (BSP_CFG_RTOS == 2)
```

```
    BaseType_t err_queue = pdFALSE;
    BaseType_t err_task = pdFALSE;
#endif  /* (BSP_CFG_RTOS == 2) */
 /* Open USB instance */
 R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (USB_SPEED_FS == g_basic0_cfg.usb_speed)
    {
       g_max_packet_size = FS_MAX_PACKET_SIZE;
    }
 else
    {
       g_max_packet_size = HS_MAX_PACKET_SIZE;
    }
#if (BSP_CFG_RTOS == 2)
    err_task = xTaskCreate((TaskFunction_t) usb_app_interrupt_event_task,
 "INT_EVT_TSK",
                           INT_EVT_TSK_STACK_SIZE,
                           NULL,
                           INT_EVT_TSK_PRI,
                           &g_app_interrupt_event_notify_tsk_hdl);
 if (pdPASS != err_task)
    {
 return;
    }
#endif
 while (true)
    {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
     {
           handle_error(FSP_ERR_ABORTED);
      }
#if (BSP_CFG_RTOS == 2)
 /* Receive message from queue */
```

```
      err_queue = xQueueReceive(g_event_queue, &p_usb_event, (portMAX_DELAY));
 /* Handle error */
 if (pdTRUE != err_queue)
     {
           handle_error(FSP_ERR_ABORTED);
      }
 if (pdTRUE == err_queue)
     {
 if (p_usb_event == &g_event_info2)
     {
 /* process Application Interrupt events */
               err = process_usb_events_for_interrupt();
               handle_error(err);
           }
 else
     {
 /* process USB events */
               err = process_usb_events();
               handle_error(err);
           }
      }
#else  /* (BSP_CFG_RTOS == 2) */
 R_USB_EventGet(&g_usb_event, &g_event);
 if (USB_STATUS_NONE == g_event)
     {
 /* Interrupt IN Test Event for Vendor Class */
 if (1 == g_interrupt_in_test_flag)
     {
             g_interrupt_in_test_flag = 0;
               g_event2                = USB_STATUS_VENDOR_INTIN_TEST;
               err = process_usb_events_for_interrupt();
           }
 /* Interrupt OUT Test Event for Vendor Class */
 else if (1 == g_interrupt_out_test_flag)
```

```
     {
             g_interrupt_out_test_flag = 0;
               g_event2 = USB_STATUS_VENDOR_INTOUT_TEST;
               err      = process_usb_events_for_interrupt();
           }
 /* Interrupt IN Test Event for COM Class */
 else if (0 != g_interrupt_com_event)
     {
             g_interrupt_com_event = 0;
               g_event2             = USB_STATUS_COM_IN_TEST;
               err = process_usb_events_for_interrupt();
           }
 /* ReInit Test Event */
 else if (0 != g_test_reinit_flag)
     {
             g_test_reinit_flag = 0;
               g_event2          = USB_STATUS_RE_INIT_TEST;
               err                = process_usb_events_for_interrupt();
           }
 else
     {
               err = process_usb_events();
           }
      }
 else
     {
           p_usb_event = &g_usb_event;
           err         = process_usb_events();
      }
      handle_error(err);
#endif
    }
}
/*****************************************************************************
```

```
 * End of function usb_composite_thread_entry
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : process_usb_events
 * Description : Function processes usb events.
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 ***********************************************************************************/
static fsp_err_t process_usb_events (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* USB event received */
#if (BSP_CFG_RTOS == 2)
 switch (p_usb_event->event)
#else  /* (BSP_CFG_RTOS == 2) */
 switch (g_event)
#endif /* (BSP_CFG_RTOS == 2) */
    {
 case USB_STATUS_CONFIGURED:    /* Configured State */
     {
 /* Process USB configured event */
           usb_configured_event_process();
 /* [PCDC] Read data from terminal software */
 R_USB_Read(&g_basic0_ctrl, g_cdc_buf, g_max_packet_size, USB_CLASS_PCDC);
 /* [PVND] Read data from ra_usb_hvnd.exe */
           memset(g_vnd_buf, RESET_VALUE, BUF_SIZE);
           err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], BUF_SIZE,
g_apl_vendor_bulk_out);
 break;
      }
 case USB_STATUS_WRITE_COMPLETE: /* Write Complete State */
     {
 if (FSP_SUCCESS == p_usb_event->status)
```

```
     {
 if (b_usb_attach)
     {
 /* [PCDC] Read data from terminal software */
 if ((USB_CLASS_PCDC == p_usb_event->type) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
     {
                      err = R_USB_Read(&g_basic0_ctrl, g_cdc_buf,
g_max_packet_size, USB_CLASS_PCDC);
             }
 /* [PVND] Read data from ra_usb_hvnd.exe */
 if ((g_apl_vendor_bulk_in == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
     {
             memset(g_vnd_buf, RESET_VALUE, BUF_SIZE);
 /* Read data back */
             err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], BUF_SIZE,
g_apl_vendor_bulk_out);
             }
             }
 else
     {
 // Do Nothing as USB is removed and not connected yet.
             }
           }
 break;
      }
 case USB_STATUS_READ_COMPLETE: /* Read Complete State */
     {
 if ((FSP_SUCCESS == p_usb_event->status) || (FSP_ERR_USB_SIZE_SHORT ==
p_usb_event->status))
     {
 if (b_usb_attach)
     {
```

```
 /* PCDC */
 if ((USB_CLASS_PCDC == p_usb_event->type) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
     {
 /* Write back the read data from terminal software to it. */
                       err = R_USB_Write(&g_basic0_ctrl, g_cdc_buf,
p_usb_event->data_size, USB_CLASS_PCDC);
             }
 /* PVND */
 if ((g_apl_vendor_bulk_out == p_usb_event->pipe) && (FSP_ERR_USB_FAILED !=
p_usb_event->status))
     {
 /* Data comparison read from host */
                       err = buffer_check(p_usb_event->data_size);
 if (FSP_SUCCESS == err)
     {
 /* Write data back to host */
 R_USB_PipeWrite(&g_basic0_ctrl,
                                          &g_vnd_buf[RESET_VALUE],
                                          p_usb_event->data_size,
                                          g_apl_vendor_bulk_in);
             }
 else
     {
 return FSP_ERR_USB_FAILED;
             }
             }
             }
 else
     {
 // Do Nothing as USB is removed and not connected yet.
             }
           }
 break;
```

```
      }
case USB_STATUS_REQUEST:       /* Receive Class Request */
     {
/* Perform usb status request operation.*/
           err = usb_status_request();
break;
      }
case USB_STATUS_REQUEST_COMPLETE: /* Request Complete State */
     {
// Do Nothing.
break;
      }
case USB_STATUS_DETACH:
case USB_STATUS_SUSPEND:
     {
/* Reset the usb attached flag as indicating usb is removed.*/
           b_usb_attach = false;
           memset(g_cdc_buf, RESET_VALUE, sizeof(g_cdc_buf));
           memset(g_vnd_buf, RESET_VALUE, sizeof(g_vnd_buf));
break;
      }
case USB_STATUS_RESUME:
     {
/* set the usb attached flag*/
           b_usb_attach = true;
break;
      }
default:
     {
break;
      }
   }
return err;
}
```

```
/***********************************************************************************
 * End of function process_usb_events
 ***********************************************************************************/
/***********************************************************************************
 * Function Name : process_usb_events_for_interrupt
 * Description : Function processes Application events (Interrupt IN/OUT Request ).
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 ***********************************************************************************/
static fsp_err_t process_usb_events_for_interrupt (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint8_t   pipe = RESET_VALUE;
 /* USB event received */
#if (BSP_CFG_RTOS == 2)
 switch (p_usb_event->event)
#else  /* (BSP_CFG_RTOS == 2) */
 switch (g_event2)
#endif /* (BSP_CFG_RTOS == 2) */
    {
 case USB_STATUS_VENDOR_INTIN_TEST:
     {
           pipe = g_apl_vendor_interrupt_in;
           err = R_USB_PipeWrite(&g_basic0_ctrl, (uint8_t *) "interrupt-in-test",
strlen("interrupt-in-test"), pipe);
 break;
      }
 case USB_STATUS_VENDOR_INTOUT_TEST:
     {
           pipe = g_apl_vendor_interrupt_out;
           err = R_USB_PipeRead(&g_basic0_ctrl, &g_vnd_buf[RESET_VALUE], (BUF_SIZE),
pipe);
 if (FSP_SUCCESS == err)
```

```
     {
                g_interrupt_out_times++;
            }
 break;
       }
 case USB_STATUS_COM_IN_TEST:
     {
            pipe = g_apl_cdc_interrupt_in;
            err = R_USB_PipeWrite(&g_basic0_ctrl, g_interrupt_in_com_data,
sizeof(g_interrupt_in_com_data), pipe);
 break;
       }
 case USB_STATUS_RE_INIT_TEST:
     {
            g_usb_on_usb.close(&g_basic0_ctrl);
     g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 break;
       }
 default:
     {
 break;
       }
   }
 return err;
}
/******************************************************************************
 * End of function process_usb_events_for_interrupt
 ******************************************************************************/
/******************************************************************************
 * Function Name : usb_configured_event_process
 * Description : Function processes USB configured event (for vendor class).
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
```

```
 ***********************************************************************************/
static fsp_err_t usb_configured_event_process (void)
{
 fsp_err_t err       = FSP_SUCCESS;
    uint16_t   used_pipe = RESET_VALUE;
    usb_pipe_t pipe_info = {RESET_VALUE};
    uint8_t    pipe      = RESET_VALUE;
 /* Get USB Pipe Information */
    err = R_USB_UsedPipesGet(&g_basic0_ctrl, &used_pipe, USB_CLASS_PVND);
 if (FSP_SUCCESS == err)
    {
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
     {
 /* check for the used pipe */
 if ((used_pipe & (START_PIPE << pipe)) != RESET_VALUE)
     {
 /* Get the pipe Info */
               err = R_USB_PipeInfoGet(&g_basic0_ctrl, &pipe_info, pipe);
 /* Mask out the direction bit to obtain the endpoint numb value for switch
evaluation */
 switch (pipe_info.endpoint & ~USB_EP_DIR_IN)
     {
 case USB_VENDOR_BULK_OUT_EP:
     {
                     g_apl_vendor_bulk_out = pipe;
 break;
             }
 case USB_VENDOR_BULK_IN_EP:
     {
                     g_apl_vendor_bulk_in = pipe;
 break;
             }
 case USB_VENDOR_INT_OUT_EP:
     {
```

```
                        g_apl_vendor_interrupt_out = pipe;
 break;
             }
 case USB_VENDOR_INT_IN_EP:
     {
                        g_apl_vendor_interrupt_in = pipe;
 break;
             }
 case USB_CDC_INT_IN_EP:
     {
                        g_apl_cdc_interrupt_in = pipe;
 break;
             }
 default:
     {
 /* Do nothing */
 break;
             }
             }
           }
 else
     {
 /* Do nothing */
           }
      }
   }
 return err;
}
/******************************************************************************
 * End of function usb_configured_event_process
 ******************************************************************************/
/******************************************************************************
 * Function Name : usb_status_request
 * Description : Function processes usb status request.
```

```
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 ***********************************************************************************/
static fsp_err_t usb_status_request (void)
{
 fsp_err_t err            = FSP_SUCCESS;
    uint16_t request_length = RESET_VALUE;
 /* Check for the specific CDC class request IDs */
 if (USB_PCDC_SET_LINE_CODING == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
 /* Get the class request.*/
       err = R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
    }
 else if (USB_PCDC_GET_LINE_CODING == (p_usb_event->setup.request_type & USB_BREQUEST
))
    {
 /* Set the class request.*/
       err = R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
    }
 else if (USB_PCDC_SET_CONTROL_LINE_STATE == (p_usb_event->setup.request_type &
USB_BREQUEST))
    {
 /* Set the usb status as ACK response.*/
       err = R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
    }
 else if (USB_SET_VENDOR_NO_DATA == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
 /* Set ACk to host */
       err = R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
    }
 else if (USB_SET_VENDOR == (p_usb_event->setup.request_type & USB_BREQUEST))
```

```
    {
       request_length = p_usb_event->setup.request_length;
 /* Get data length from host */
       err = R_USB_PeriControlDataGet(&g_basic0_ctrl, &g_request_buf[RESET_VALUE],
request_length);
    }
 else if (USB_GET_VENDOR == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
       request_length = p_usb_event->setup.request_length;
 /* Set data length in peripheral */
       err = R_USB_PeriControlDataSet(&g_basic0_ctrl, &g_request_buf[RESET_VALUE],
request_length);
    }
 else
    {
 // Do Nothing.
    }
 return err;
}
/******************************************************************************
 * End of function usb_status_request
 ******************************************************************************/
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_composite_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_event_info : Control structure for USB API.
 * usb_hdl_t handler : Task Handle
 * usb_onoff_t usbon_off_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_composite_callback (usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t on_off)
{
```

```
 FSP_PARAMETER_NOT_USED(handler);
 FSP_PARAMETER_NOT_USED(on_off);
 /* Send event received to queue */
 if (pdTRUE != (xQueueSend(g_event_queue, (const void *) &p_event_info, (TickType_t)
(RESET_VALUE))))
    {
       g_err_flag = true;
    }
}
/******************************************************************************
 * End of function usb_composite_callback
 ******************************************************************************/
#endif  /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : handle_error
 * Description : Closes the USB module , Print and traps error.
 * Arguments : fsp_err_t err : error status
 * Return value : none
 ******************************************************************************/
static void handle_error (fsp_err_t err)
{
 if (FSP_SUCCESS != err)
    {
 R_USB_Close(&g_basic0_ctrl);
    }
}
/******************************************************************************
 * End of function handle_error
 ******************************************************************************/
/******************************************************************************
 * Function Name : buffer_check
 * Description : Check data received from vendor host tools
 * Arguments : uint32_t length : data length
 * char * err_str : error string
```

```
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation.
 ***********************************************************************************/
static fsp_err_t buffer_check (uint32_t length)
{
 for (uint16_t cnt = RESET_VALUE; cnt < (uint16_t) length; cnt++)
    {
 if ((uint8_t) (cnt & USB_VALUE_FF) != g_vnd_buf[cnt])
      {
 return FSP_ERR_ABORTED;
       }
    }
 R_BSP_SoftwareDelay(DELAY, BSP_DELAY_UNITS_MILLISECONDS);
 return FSP_SUCCESS;
}
/***********************************************************************************
 * End of function buffer_check
 ***********************************************************************************/
```

### 5.2.6.35 USB HAUD (r_usb_haud)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_USB_HAUD_DeviceInfoGet (usb_ctrl_t *const p_api_ctrl, usb_haud_device_info_t *p_info, uint8_t device_address)<br>Check audio version of connected device. More... |
| fsp_err_t | R_USB_HAUD_AlternateNumberGet (usb_ctrl_t *const p_api_ctrl, usb_haud_sampling_t *p_info, uint8_t direction, uint8_t device_address)<br>Get the alternate number for SET_INTERFACE. (Audio 1.0 support only) More... |
| fsp_err_t | R_USB_HAUD_PipeSet (usb_ctrl_t *const p_api_ctrl, uint32_t interface_number, uint32_t alternate_number, uint8_t device_address) |

| | |
|---|---|
| | Find the interface descriptor for streaming data transfer and configure the pipe. More... |

### Detailed Description

This module provides a USB Peripheral Audio Device Class Driver (HAUD). It implements the USB HAUD Interface.

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_haud module combines with the r_usb_basic module to provide USB Host Audio Device Class (HAUD) driver.

### Features

The r_usb_haud module has the following key features:

- Data transfer to and from USB peripheral audio device.
- Transmissio Audio class request to USB peripheal audio device.
- Supports Audio 1.0

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

- Audio 1.0

| Request | Code | Description |
|---|---|---|
| SET_CUR | 0x01 | Current setting attribute |
| GET_CUR | 0x81 | Current setting attribute |

Note:

For details about each device class, please refer to Audio Class specification.

### Limitations

- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

# Examples

### USB HAUD Example

1. Example code for Audio 1.0

```
#define AUDIO_BUFFER_SIZE (3 * 1024) + 1024
#define REQUEST_SIZE (3 * 1024)
#define FIRST_PACKET_TEST_SIZE (200) /* for Test Device (USBX-PAUD Sample Project :
EK-RA6M5 Board) */
 /* The first 200 bytes of the loopback data are all 0x00.      */
#define READ_REQUEST_SIZE (REQUEST_SIZE + FIRST_PACKET_TEST_SIZE)
#define BMREQUESTTYPEDIR (0x0080U)
#define BMREQUESTTYPETYPE (0x0060U)
#define ALT_NUM_0 (0)
#define SAMPLING_FREQUENCY_CHANNNELS (2)
#define SAMPLING_FREQUENCY_FREQUENCY (48000)
#define SAMPLING_FREQUENCY_RESOLUTION (16)
#define SAMPLING_FREQUENCY_DATA_SIZE (3)
/******************************************************************************
 * Exported global functions (to be accessed by other files)
 ******************************************************************************/
void usb_haud_example(void);
#if (BSP_CFG_RTOS == 2)
void usb_apl_callback(usb_event_info_t * p_event_info, usb_hdl_t cur_task,
usb_onoff_t usb_state);
#endif
/******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
```

```
#if (BSP_CFG_RTOS == 2)
static volatile usb_event_info_t * p_usb_event = NULL;
extern QueueHandle_t g_usb_queue;
#else
static usb_event_info_t event_info = {0};
static volatile usb_event_info_t * p_usb_event = &event_info;
#endif
/* Private variables */
static uint8_t g_buf[8] BSP_ALIGN_VARIABLE(4) = {RESET_VALUE}; // Class request data
buffer
static uint8_t g_send_buffer[AUDIO_BUFFER_SIZE] BSP_ALIGN_VARIABLE(4);
static uint8_t g_recv_buffer[AUDIO_BUFFER_SIZE] BSP_ALIGN_VARIABLE(4);
static uint8_t g_usb_dummy = RESET_VALUE; // dummy variable to send
static volatile bool g_err_flag = false; // flag bit
static uint8_t g_device_address = 0;
static uint32_t g_completed_read_transfer_length_total = 0;
static usb_haud_sampling_t g_sampling_tx;
static usb_haud_sampling_t g_sampling_rx;
/* Private functions */
static void haud_app_data_buffer_init(void);
static void haud_app_standard_request_complete(void);
static void haud_app_audio_class_request_complete(void);
static fsp_err_t haud_app_interface_set(uint32_t value, uint32_t index);
static fsp_err_t haud_app_sampling_frequency_set(uint8_t ep_address, uint32_t
frequency);
static fsp_err_t haud_app_sampling_frequency_get(uint8_t ep_address);
static void handle_error(fsp_err_t err, char * err_str);
extern const usb_cfg_t g_basic0_cfg;
extern usb_instance_ctrl_t g_basic0_ctrl;
/**********************************************************************************
*****************************
 * Function Name : usb_haud_example
 * Description : Host Audio application main process
 * Arguments : none
```

```
 * Return value : none
 ***********************************************************************************************
*******************************/
void usb_haud_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
#if (BSP_CFG_RTOS == 2)
    BaseType_t err_queue = USB_FALSE;
#endif
 usb_haud_device_info_t device_info;
 fsp_pack_version_t version = {RESET_VALUE};
 /* version get API for FLEX pack information */
 R_FSP_VersionGet(&version);
    err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Error Handle */
 if (FSP_SUCCESS != err)
    {
       APP_ERR_TRAP
    }
    haud_app_data_buffer_init();
 while (true)
    {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
     {
           handle_error(FSP_ERR_ABORTED, "Error in sending usb event through
queue");
     }
#if (BSP_CFG_RTOS == 2)
 /* Receive message from queue and analyzing the received message*/
      err_queue = xQueueReceive(g_usb_queue, &p_usb_event, (portMAX_DELAY));
 /* Handle error */
 if (USB_TRUE != err_queue)
     {
```

```
            handle_error(FSP_ERR_ABORTED, "Error in receiving USB event message
through queue");
      }
#else
 /* Obtain USB related events */
       err = R_USB_EventGet(&event_info, &event_info.event);
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
#endif
 switch (p_usb_event->event)
      {
 case USB_STATUS_CONFIGURED:
      {
                g_device_address = p_usb_event->device_address;
 /* Get device information (Audio version) */
                err = R_USB_HAUD_DeviceInfoGet(&g_basic0_ctrl, &device_info,
g_device_address);
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
 if (USB_HAUD_PROTOCOL_IP_VERSION_01_00 == device_info.protocol)
      {
                   g_sampling_tx.channels   = SAMPLING_FREQUENCY_CHANNNELS;
                   g_sampling_tx.frequency = SAMPLING_FREQUENCY_FREQUENCY;
                   g_sampling_tx.resolution = SAMPLING_FREQUENCY_RESOLUTION;
 /* Get Alternate number for Isochronous-OUT IF */
                   err =
 R_USB_HAUD_AlternateNumberGet(&g_basic0_ctrl, &g_sampling_tx, USB_EP_DIR_OUT,
g_device_address);
 if (FSP_SUCCESS != err)
```

```
     {
        APP_ERR_TRAP
     }
 /* SET_INTERFACE (Use the acquired ALT number) */
                  err = haud_app_interface_set(g_sampling_tx.alternate_number,
g_sampling_tx.interface_number);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
 else
     {
 /* Do nothing. (Audio 2.0 and later are not supported.) */
     }
 break;
     }
 case USB_STATUS_READ_COMPLETE:
     {
 if ((FSP_SUCCESS == p_usb_event->status) || (FSP_ERR_USB_SIZE_SHORT ==
p_usb_event->status))
     {
                  g_completed_read_transfer_length_total += p_usb_event->data_size;
 if (READ_REQUEST_SIZE > g_completed_read_transfer_length_total)
     {
                    err = R_USB_Read(&g_basic0_ctrl,
                                     g_recv_buffer +
g_completed_read_transfer_length_total,
                                     g_sampling_rx.max_packet_size,
                                     g_device_address);
 /* Handle Error */
 if (FSP_SUCCESS != err)
     {
                          handle_error(err, "**R_USB_Read API FAILED**");
```

```
     }
     }
 else
     {
 if (0 == memcmp(g_send_buffer, g_recv_buffer + FIRST_PACKET_TEST_SIZE,
REQUEST_SIZE))
     {
 /* SET_INTERFACE (ALT number : 0) */
                    err = haud_app_interface_set(ALT_NUM_0,
g_sampling_rx.interface_number);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
 else
     {
        APP_ERR_TRAP
     }
     }
     }
 else
     {
        APP_ERR_TRAP
     }
 break;
     }
 case USB_STATUS_WRITE_COMPLETE:
     {
 if (FSP_SUCCESS == p_usb_event->status)
     {
 /* SET_INTERFACE (ALT number : 0) */
                   err = haud_app_interface_set(ALT_NUM_0,
g_sampling_tx.interface_number);
```

```
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
else
     {
        APP_ERR_TRAP
     }
break;
     }
case USB_STATUS_REQUEST_COMPLETE:
     {
switch (BMREQUESTTYPETYPE & p_usb_event->setup.request_type)
     {
case USB_STANDARD:
     {
                       haud_app_standard_request_complete();
break;
     }
case USB_CLASS:
     {
                       haud_app_audio_class_request_complete();
break;
     }
     }
break;
     }
case USB_STATUS_DETACH:
     {
               memset(&g_sampling_tx, 0, sizeof(g_sampling_tx));
               memset(&g_sampling_rx, 0, sizeof(g_sampling_rx));
               g_device_address = 0;
               g_completed_read_transfer_length_total = 0;
```

```
 break;
      }
 default:
      {
 /* No operation to do */
 break;
      }
      }
    }
} /* End of function usb_haud_example */
/*********************************************************************************
*****************************
 * Function Name : haud_app_data_buffer_init
 * Description : Initializes test write data.
 * Arguments : none
 * Return value : none
 ********************************************************************************
*****************************/
static void haud_app_data_buffer_init (void)
{
    uint32_t i = 0;
    uint32_t j = 0;
 /* Set data to the send buffer. */
 for ( ; i < AUDIO_BUFFER_SIZE; i++)
    {
 if (i % (VALUE_FFH + 1))
      {
            (g_send_buffer)[i] = (uint8_t) i;
      }
 else
      {
            (g_send_buffer)[i] = (uint8_t) j;
            j++;
      }
```

```
    }
} /* End of function haud_app_data_buffer_init */
/***********************************************************************************
*****************************
 * Function Name : haud_app_standard_request_complete
 * Description : Standard request (such as SET_INTERFACE) completion event
processing.
 * Arguments : none
 * Return value : none
 ***********************************************************************************
*****************************/
static void haud_app_standard_request_complete (void)
{
 fsp_err_t err = FSP_SUCCESS;
 if (USB_SET_INTERFACE == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
 if ((g_sampling_tx.interface_number == p_usb_event->setup.request_index) &&
            (g_sampling_tx.alternate_number == p_usb_event->setup.request_value))
     {
            g_sampling_tx.alternate_setting = USB_TRUE;
            err = R_USB_HAUD_PipeSet(&g_basic0_ctrl,
                                     g_sampling_tx.interface_number,
                                     g_sampling_tx.alternate_number,
                                     g_device_address);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
            g_sampling_tx.frequency_setting = USB_TRUE;
 /* Check FormatTypeI::bSamFreqType@7, tSamFreq@8, tSamFreq@11. */
 /* If there is only one frequency, no need to set.             */
 if ((g_sampling_tx.p_sampling_descriptor[7] == 1) ||
               ((g_sampling_tx.p_sampling_descriptor[7] == 0) &&
               (g_sampling_tx.
```

```
                  p_sampling_descriptor[8] ==
g_sampling_tx.p_sampling_descriptor[11]) &&
                (g_sampling_tx.
                  p_sampling_descriptor[9] ==
g_sampling_tx.p_sampling_descriptor[12]) &&
                (g_sampling_tx.
                  p_sampling_descriptor[10] ==
g_sampling_tx.p_sampling_descriptor[13])))
     {
                g_sampling_tx.frequency_setting = USB_FALSE;
     }
 if (g_sampling_tx.frequency_setting)
     {
 /* Send SET_CUR requests to set frequency. */
                err = haud_app_sampling_frequency_set(g_sampling_tx.endpoint_address,
g_sampling_tx.frequency);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
     }
 else if ((g_sampling_tx.interface_number == p_usb_event->setup.request_index) &&
                (ALT_NUM_0 == p_usb_event->setup.request_value))
     {
           g_sampling_tx.alternate_setting = USB_FALSE;
           g_sampling_rx.channels   = SAMPLING_FREQUENCY_CHANNNELS;
           g_sampling_rx.frequency = SAMPLING_FREQUENCY_FREQUENCY;
           g_sampling_rx.resolution = SAMPLING_FREQUENCY_RESOLUTION;
 /* Get Alternate number for Isochronous-IN IF */
           err = R_USB_HAUD_AlternateNumberGet(&g_basic0_ctrl, &g_sampling_rx,
USB_EP_DIR_IN, g_device_address);
 if (FSP_SUCCESS != err)
     {
```

```
        APP_ERR_TRAP
     }
 /* SET_INTERFACE (Use the acquired ALT number) */
           err = haud_app_interface_set(g_sampling_rx.alternate_number,
g_sampling_rx.interface_number);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
 else if ((g_sampling_rx.interface_number == p_usb_event->setup.request_index) &&
               (g_sampling_rx.alternate_number == p_usb_event->setup.request_value))
     {
           g_sampling_rx.alternate_setting = USB_TRUE;
           err = R_USB_HAUD_PipeSet(&g_basic0_ctrl,
                                    g_sampling_rx.interface_number,
                                    g_sampling_rx.alternate_number,
                                    g_device_address);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
           g_sampling_rx.frequency_setting = USB_TRUE;
 /* Check FormatTypeI::bSamFreqType@7, tSamFreq@8, tSamFreq@11. */
 /* If there is only one frequency, no need to set.             */
 if ((g_sampling_rx.p_sampling_descriptor[7] == 1) ||
               ((g_sampling_rx.p_sampling_descriptor[7] == 0) &&
               (g_sampling_rx.
                 p_sampling_descriptor[8] ==
g_sampling_rx.p_sampling_descriptor[11]) &&
               (g_sampling_rx.
                 p_sampling_descriptor[9] ==
g_sampling_rx.p_sampling_descriptor[12]) &&
               (g_sampling_rx.
```

```
                  p_sampling_descriptor[10] ==
g_sampling_rx.p_sampling_descriptor[13])))
     {
                g_sampling_rx.frequency_setting = USB_FALSE;
     }
 if (g_sampling_rx.frequency_setting)
     {
 /* Send SET_CUR requests to set frequency. */
                err = haud_app_sampling_frequency_set(g_sampling_rx.endpoint_address,
g_sampling_rx.frequency);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
     }
 else if ((g_sampling_rx.interface_number == p_usb_event->setup.request_index) &&
                (ALT_NUM_0 == p_usb_event->setup.request_value))
     {
           g_sampling_rx.alternate_setting = USB_FALSE;
     }
 else
     {
        APP_ERR_TRAP
     }
   }
} /* End of function haud_app_standard_request_complete */
/***********************************************************************************
*****************************
 * Function Name : haud_app_audio_class_request_complete
 * Description : Audio class request completion event processing.
 * Arguments : none
 * Return value : none
 ***********************************************************************************
```

```
*******************************/
static void haud_app_audio_class_request_complete (void)
{
    uint32_t value;
 fsp_err_t err = FSP_SUCCESS;
 /* SET_CUR (Isochronous-OUT IF's EP) completed */
 if ((USB_HAUD_AUDIO10_SET_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
&&
       (g_sampling_tx.endpoint_address == p_usb_event->setup.request_index))
    {
 /* GET_CUR Request (Check whether the sampling frequency specified by SET_CUR has
been set on the device.) */
       err = haud_app_sampling_frequency_get(g_sampling_tx.endpoint_address);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
    }
 /* SET_CUR (Isochronous-IN IF's EP) completed */
 else if ((USB_HAUD_AUDIO10_SET_CUR == (p_usb_event->setup.request_type &
USB_BREQUEST)) &&
            (g_sampling_rx.endpoint_address == p_usb_event->setup.request_index))
    {
 /* GET_CUR Request (Check whether the sampling frequency specified by SET_CUR has
been set on the device.) */
       err = haud_app_sampling_frequency_get(g_sampling_rx.endpoint_address);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
    }
 /* GET_CUR (Isochronous-OUT IF's EP) completed */
 else if ((USB_HAUD_AUDIO10_GET_CUR == (p_usb_event->setup.request_type &
USB_BREQUEST)) &&
```

```
        (g_sampling_tx.endpoint_address == p_usb_event->setup.request_index))
   {
 /* The fourth byte has no data so it is cleared. */
      g_buf[3] = 0;
      value = (ULONG) g_buf[0];
      value |= (ULONG) g_buf[1] << 8;
      value |= (ULONG) g_buf[2] << 16;
      value |= (ULONG) g_buf[3] << 24;
 if (value != g_sampling_tx.frequency)
     {
 /* Sampling frequency requested is not set to device. */
        APP_ERR_TRAP
     }
 /* Data Write (Isochronous-OUT) */
      err =
 R_USB_Write(&g_basic0_ctrl, (uint8_t *) g_send_buffer, REQUEST_SIZE,
g_device_address);
 if (FSP_SUCCESS != err)
     {
           handle_error(err, "**R_USB_Write API FAILED**");
     }
   }
 /* GET_CUR (Isochronous-IN IF's EP) completed */
 else if ((USB_HAUD_AUDIO10_GET_CUR == (p_usb_event->setup.request_type &
USB_BREQUEST)) &&
            (g_sampling_rx.endpoint_address == p_usb_event->setup.request_index))
   {
 /* The fourth byte has no data so it is cleared. */
      g_buf[3] = 0;
      value = (ULONG) g_buf[0];
      value |= (ULONG) g_buf[1] << 8;
      value |= (ULONG) g_buf[2] << 16;
      value |= (ULONG) g_buf[3] << 24;
 if (value != g_sampling_rx.frequency)
```

```
     {
 /* Sampling frequency requested is not set to device. */
        APP_ERR_TRAP
     }
 /* Data Read (Isochronous-IN) */
      err = R_USB_Read(&g_basic0_ctrl, g_recv_buffer, g_sampling_rx.max_packet_size,
g_device_address);
 /* Handle Error */
 if (FSP_SUCCESS != err)
     {
           handle_error(err, "**R_USB_Read API FAILED**");
     }
   }
 else
   {
      APP_ERR_TRAP
   }
} /* End of function haud_app_audio_class_request_complete */
/**********************************************************************************
*****************************
 * Function Name : haud_app_interface_set
 * Description : This function does a SET_INTERFACE with the specified Alternate
number and interface number.
 * Arguments : uint32_t value : Alternate number
 * : uint32_t index : Interface number
 * Return value : fsp_err_t err : Result of a control transfer request
 **********************************************************************************
*****************************/
static fsp_err_t haud_app_interface_set (uint32_t value, uint32_t index)
{
 fsp_err_t   err = FSP_SUCCESS;
   usb_setup_t setup;
   setup.request_type   = USB_SET_INTERFACE | USB_HOST_TO_DEV | USB_STANDARD |
USB_INTERFACE;
```

```
    setup.request_value = (uint16_t) value;
    setup.request_index = (uint16_t) index;
    setup.request_length = 0;
    err = R_USB_HostControlTransfer(&g_basic0_ctrl, &setup, &g_usb_dummy,
g_device_address);
 return err;
} /* End of function haud_app_interface_set */
/***********************************************************************************
*****************************
 * Function Name : haud_app_sampling_frequency_set
 * Description : This function sets the sampling frequency on the device using the
SET_CUR request.
 * Arguments : uint8_t ep_address : End Point Address
 * : uint32_t frequency : Sampling Furequency
 * Return value : fsp_err_t err : Result of a control transfer request
 ***********************************************************************************
*****************************/
static fsp_err_t haud_app_sampling_frequency_set (uint8_t ep_address, uint32_t
frequency)
{
 fsp_err_t   err = FSP_SUCCESS;
    usb_setup_t setup;
    setup.request_type   = USB_HAUD_AUDIO10_SET_CUR | USB_HOST_TO_DEV | USB_CLASS |
USB_ENDPOINT;
    setup.request_value = USB_HAUD_AUDIO10_EP_SAMPLING_FREQ_CONTROL << 8;
    setup.request_index = (uint16_t) ep_address;
    setup.request_length = SAMPLING_FREQUENCY_DATA_SIZE;
    g_buf[0] = (uint8_t) frequency;
    g_buf[1] = (uint8_t) (frequency >> 8);
    g_buf[2] = (uint8_t) (frequency >> 16);
    g_buf[3] = 0;
 /* Request Control transfer */
    err = R_USB_HostControlTransfer(&g_basic0_ctrl, &setup, g_buf, g_device_address);
 return err;
```

```
} /* End of function haud_app_sampling_frequency_set */
/***********************************************************************************
*******************************
 * Function Name : haud_app_sampling_frequency_get
 * Description : This function uses a GET_CUR request to get the sampling frequency
of the device.
 * Arguments : uint8_t ep_address : End Point Address
 * Return value : fsp_err_t err : Result of a control transfer request
 ***********************************************************************************
*******************************/
static fsp_err_t haud_app_sampling_frequency_get (uint8_t ep_address)
{
 fsp_err_t   err = FSP_SUCCESS;
    usb_setup_t setup;
    setup.request_type   = USB_HAUD_AUDIO10_GET_CUR | USB_DEV_TO_HOST | USB_CLASS |
USB_ENDPOINT;
    setup.request_value = USB_HAUD_AUDIO10_EP_SAMPLING_FREQ_CONTROL << 8;
    setup.request_index = ep_address;
    setup.request_length = SAMPLING_FREQUENCY_DATA_SIZE;
    err = R_USB_HostControlTransfer(&g_basic0_ctrl, &setup, g_buf, g_device_address);
 return err;
} /* End of function haud_app_sampling_frequency_get */
/***********************************************************************************
*******************************
 * Function Name : handle_error
 * Description : Print and traps error
 * Arguments : fsp_err_t err
 * : char * err_str
 * Return value : none
 ***********************************************************************************
*******************************/
static void handle_error (fsp_err_t err, char * err_str)
{
 fsp_err_t error = FSP_SUCCESS;
```

```
 FSP_PARAMETER_NOT_USED(err);
 FSP_PARAMETER_NOT_USED(err_str);
 /* close opened USB module */
    error = R_USB_Close(&g_basic0_ctrl);
 /* Handle error */
 if (FSP_SUCCESS != error)
    {
       APP_ERR_TRAP
    }
} /* End of function handle_error() */
#if (BSP_CFG_RTOS == 2)
/**********************************************************************************
*******************************
 * Function Name : usb_apl_callback
 * Description : This function is callback for USB
 * Arguments : usb_event_info_t *p_event_info
 * : usb_hdl_t handler
 * : usb_onoff_t usb_state
 * Return value : none
 **********************************************************************************
*******************************/
void usb_apl_callback (usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t usb_state)
{
 FSP_PARAMETER_NOT_USED(handler);
 FSP_PARAMETER_NOT_USED(usb_state);
 /* Send event received to queue */
 if (USB_TRUE != (xQueueSend(g_usb_queue, (const void *) &p_event_info, (TickType_t)
(NO_WAIT_TIME))))
    {
       g_err_flag = true;
    }
} /* End of function usb_rtos_callback */
#endif
```

## Function Documentation

### ◆ R_USB_HAUD_DeviceInfoGet()

fsp_err_t R_USB_HAUD_DeviceInfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_haud_device_info_t * *p_info*, uint8_t *device_address* )

Check audio version of connected device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_HAUD_AlternateNumberGet()

fsp_err_t R_USB_HAUD_AlternateNumberGet ( usb_ctrl_t *const *p_api_ctrl*, usb_haud_sampling_t * *p_info*, uint8_t *direction*, uint8_t *device_address* )

Get the alternate number for SET_INTERFACE. (Audio 1.0 support only)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_HAUD_PipeSet()

fsp_err_t R_USB_HAUD_PipeSet ( usb_ctrl_t *const *p_api_ctrl*, uint32_t *interface_number*, uint32_t *alternate_number*, uint8_t *device_address* )

Find the interface descriptor for streaming data transfer and configure the pipe.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

## 5.2.6.36 USB HCDC (r_usb_hcdc)

Modules » Connectivity

### Functions

fsp_err_t R_USB_HCDC_ControlDataRead (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address)

Read Control Data.(CDC Interrupt IN data) More...

fsp_err_t R_USB_HCDC_SpecificDeviceRegister (usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id)

Register the specified vendor class device in the device table. More...

fsp_err_t R_USB_HCDC_DeviceInfoGet (usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address)

Get the VID, PID and subclass code of the connected device. More...

### Detailed Description

This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface.

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_hcdc module, when used in combination with the r_usb_basic module, operates as a USB Host Communications Device Class (HCDC) driver. The HCDC conforms to the PSTN device subclass abstract control model of the USB Communications Device Class (CDC) specification and enables communication with a CDC peripheral device.

### Features

The r_usb_hcdc module has the following key features:

- Checks for connected devices
- Implementation of communication line settings
- Acquisition of the communication line state
- Data transfer to and from a CDC peripheral device

# Configuration

### Build Time Configurations for r_usb_hcdc

The following build time configurations are defined in fsp_cfg/r_usb_hcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Target Peripheral Device Class ID | • CDC class supported device<br>• Vendor class device | CDC class supported device | Specify the device class ID of the CDC device to be connected. |
| USB Hub Multi CDC | • Disabled<br>• Enabled | Disabled | Enable or disable HUB support for CDC-CDC. |

### Configurations for Connectivity > USB HCDC (r_usb_hcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HCDC (r_usb_hcdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hcdc0 | Module name. |

*Note*

*Refer to the USB (r_usb_basic) module for hardware configuration options.*

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Communications Device Class (CDC), PSTN and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communications Device Class specification as defined in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2. The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections) enabling use of application programs designed for older modems.

### Basic Functions

The main functions of HCDC are the following:

- Verify connected devices
- Make communication line settings
- Acquire the communication line state
- Transfer data to and from the CDC peripheral device

### Abstract Control Model Class Requests - Host to Device

This driver supports the following class requests:

| Request | Code | Description |
|---|---|---|
| SendEncapsulatedCommand | 0x00 | Transmits an AT command as defined by the protocol used by the device (normally 0 for USB). |
| GetEncapsulatedResponse | 0x01 | Requests a response to a command transmitted by SendEncapsulatedCommand. |
| SetCommFeature | 0x02 | Enables or disables features such as device-specific 2-byte code and country setting. |
| GetCommFeature | 0x03 | Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting. |
| ClearCommFeature | 0x04 | Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting. |
| SetLineCoding | 0x20 | Makes communication line settings (communication speed, data length, parity bit, and stop bit length). |

| | | |
|---|---|---|
| GetLineCoding | 0x21 | Acquires the communication line setting state. |
| SetControlLineState | 0x22 | Makes communication line control signal (RTS, DTR) settings. |
| SendBreak | 0x23 | Transmits a break signal. |

*Note*

*For more information about Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.*

The expected data format for each command is shown below followed by dependent structures.

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SEND_ENCAPSULATED_COMMAND (0x00) | 0x0000 | 0x0000 | Data length | usb_hcdc_encapsulated_t |
| 0x21 | GET_ENCAPSULATED_RESPONSE (0x01) | 0x0000 | 0x0000 | Data length | usb_hcdc_encapsulated_t |
| 0x21 | SET_COMM_FEATURE (0x02) | usb_hcdc_feature_selector_t | 0x0000 | Data length | usb_hcdc_commfeature_t |
| 0x21 | GET_COMM_FEATURE (0x03) | usb_hcdc_feature_selector_t | 0x0000 | Data length | usb_hcdc_commfeature_t |
| 0x21 | CLEAR_COMM_FEATURE (0x04) | usb_hcdc_feature_selector_t | 0x0000 | Data length | None |
| 0x21 | SET_LINE_CODING (0x20) | 0x0000 | 0x0000 | 0x0000 | usb_hcdc_linecoding_t |
| 0xA1 | GET_LINE_CODING (0x21) | 0x0000 | 0x0000 | 0x0007 | usb_hcdc_linecoding_t |
| 0x21 | SET_CONTROL_LINE_STATE (0x22) | usb_hcdc_controllinestate_t | 0x0000 | 0x0000 | None |
| 0x21 | SEND_BREAK (0x23) | usb_hcdc_breakduration_t | 0x0000 | 0x0000 | None |

**ACM Notifications from Device to Host**

The following class notifications are supported:

| Notification | Code | Description |
|---|---|---|
| RESPONSE_AVAILABLE | 0x01 | Response to GET_ENCAPSULATED_RESPONSE |

| | | |
|---|---|---|
| SERIAL_STATE | 0x20 | Notification of serial line state |

The data types returned are as follows:

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | RESPONSE_AVAILABLE (0x01) | 0x0000 | 0x0000 | 0x0000 | None |
| 0xA1 | SERIAL_STATE (0x20) | 0x0000 | 0x0000 | 0x0002 | usb_hcdc_serialstate_t |

*Note*

*The host is notified with SERIAL_STATE whenever a change in the UART port state is detected.*

### Limitations

This driver is subject to the following limitations:

- Suspend is not supported when a data transfer is in progress. Confirm that data transfer has completed before executing suspend.
- Use of compound USB devices with CDC class support is not supported.
- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- This driver does not support Low-speed.
- This driver does not support simultaneous operation with the other device class.

# Examples

### USB HCDC Loopback Example

The main functions of the HCDC loopback example are as follows:

1. Virtual UART control settings are configured by transmitting the class request SET_LINE_CODING to the CDC device.
2. Sends receive (Bulk In transfer) requests to a CDC peripheral device and receives data.
3. Loops received data back to the peripheral by means of Bulk Out transfers.

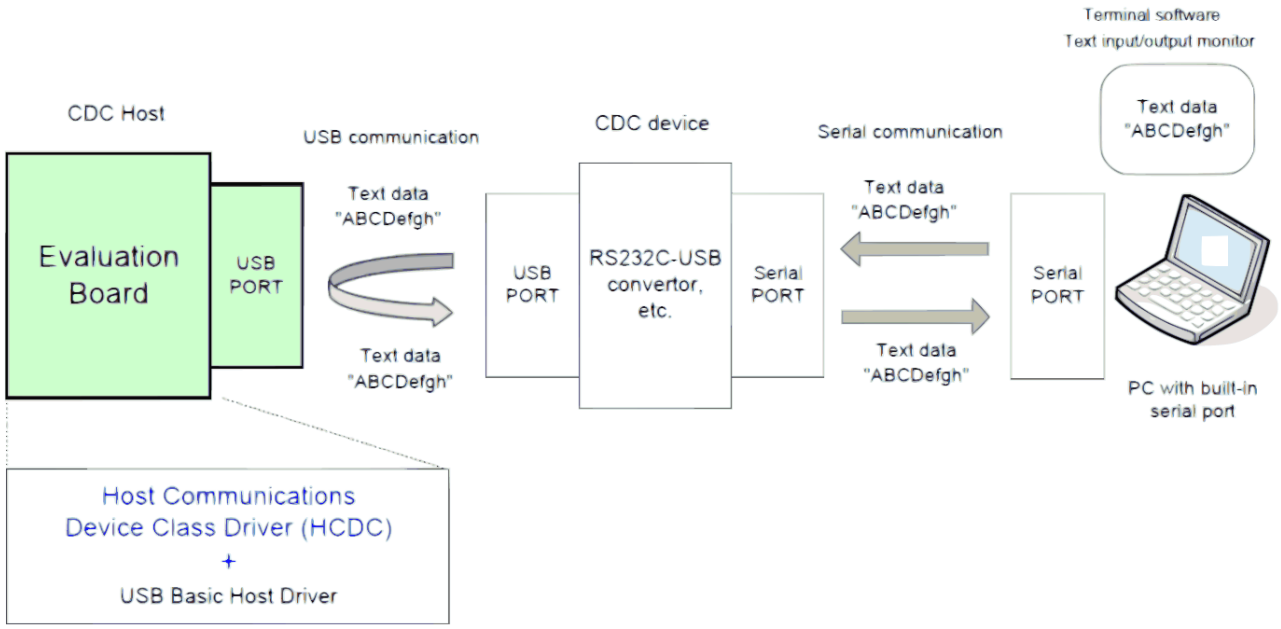



Figure 235: Data Transfer (Loopback)

The main loop performs loopback processing in which data received from a CDC peripheral device is transmitted unaltered back to the peripheral.

```
#define SET_LINE_CODING (USB_CDC_SET_LINE_CODING | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define GET_LINE_CODING (USB_CDC_GET_LINE_CODING | USB_DEV_TO_HOST | USB_CLASS |
USB_INTERFACE)
#define SET_CONTROL_LINE_STATE (USB_CDC_SET_CONTROL_LINE_STATE | USB_HOST_TO_DEV |
USB_CLASS | USB_INTERFACE)
#define COM_SPEED (9600U)
#define COM_DATA_BIT (8U)
#define COM_STOP_BIT (0)
#define COM_PARITY_BIT (0)
#define LINE_CODING_LENGTH (7)
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
```

```
 * Return value : none
 ***********************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/***********************************************************************************
 * Function Name : usb_hcdc_example
 * Description : Host CDC application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usb_hcdc_example (void)
{
 usb_status_t    event;
    usb_event_info_t event_info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
```

```
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 /* Configure virtual UART settings */
                set_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
 break;
 case USB_STATUS_READ_COMPLETE:
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
      {
 if (USB_CLASS_HCDC == event_info.type)
      {
 if (event_info.data_size > 0)
      {
 /* Send the received data back to the connected peripheral */
                           g_usb_on_usb.write(&g_basic0_ctrl, g_snd_buf,
event_info.data_size, USB_DEVICE_ADDRESS_1);
                       }
 else
      {
 /* Send the data reception request when the zero-length packet is received. */
                           g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf,
CDC_DATA_LEN, USB_DEVICE_ADDRESS_1);
                       }
                  }
 else /* USB_HCDCC */
      {
 /* Control Class notification "SerialState" receive start */
                       g_usb_on_usb.read(&g_basic0_ctrl,
                                         (uint8_t *) &g_serial_state,
                                         USB_HCDC_SERIAL_STATE_MSG_LEN,
                                         USB_DEVICE_ADDRESS_1);
```

```
                }
            }
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (FSP_SUCCESS == event_info.status)
     {
 /* Start receive operation */
                g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
            }
 break;
 case USB_STATUS_REQUEST_COMPLETE:
 if (USB_CDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* Set virtual RTS/DTR signal state */
                set_control_line_state(&g_basic0_ctrl, event_info.device_address);
/* CDC Class request "SetControlLineState" */
            }
 /* Check Complete request "SetControlLineState" */
 else if (USB_CDC_SET_CONTROL_LINE_STATE == (event_info.setup.request_type &
USB_BREQUEST))
     {
 /* Read back virtual UART settings */
                get_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
            }
 else if (USB_CDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* Now that setup is complete, start loopback operation */
                g_usb_on_usb.read(&g_basic0_ctrl, g_snd_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
            }
 else
     {
```

```
 /* Unsupported request */
                }
 break;
 default:
 /* Other event */
 break;
       }
    }
} /* End of function usb_hcdc_example() */
/******************************************************************************
 * Function Name : set_control_line_state
 * Description : Send the class request (SetControLineState) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void set_control_line_state (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = SET_CONTROL_LINE_STATE; /*
bRequestCode:SET_CONTROL_LINE_STATE, bmRequestType */
    setup.request_value = 0x0000;                /* wValue:Zero */
    setup.request_index = 0x0000;                /* wIndex:Interface */
    setup.request_length = 0x0000;                /* wLength:Zero */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_usb_dummy,
device_address);
} /* End of function set_control_line_state() */
/******************************************************************************
 * Function Name : set_line_coding
 * Description : Send the class request (SetLineCoding) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void set_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
```

```
    usb_setup_t setup;
    g_com_parm.dwdte_rate   = (usb_hcdc_line_speed_t) COM_SPEED;
    g_com_parm.bchar_format = (usb_hcdc_stop_bit_t) COM_STOP_BIT;
    g_com_parm.bparity_type = (usb_hcdc_parity_bit_t) COM_PARITY_BIT;
    g_com_parm.bdata_bits   = (usb_hcdc_data_bit_t) COM_DATA_BIT;
    setup.request_type   = SET_LINE_CODING;    /* bRequestCode:SET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function set_line_coding() */
/******************************************************************************
 * Function Name : get_line_coding
 * Description : Send the class request (GetLineCoding) to CDC device.
 * Arguments : device_address : device address of CDC device.
 * Return value : none
 ******************************************************************************/
void get_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = GET_LINE_CODING;    /* bRequestCode:GET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
} /* End of function get_line_coding() */
```

### Function Documentation

### ◆ R_USB_HCDC_ControlDataRead()

fsp_err_t R_USB_HCDC_ControlDataRead ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *device_address* )

Read Control Data.(CDC Interrupt IN data)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed (Data read request completed). |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_USB_BUSY | Data receive request already in process for USB device with same device address. |
| FSP_ERR_ASSERTION | Parameter is NULL error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*1. Do not call this API in the following function.*
*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_HCDC_SpecificDeviceRegister()

fsp_err_t R_USB_HCDC_SpecificDeviceRegister ( usb_ctrl_t *const *p_api_ctrl*, uint16_t *vendor_id*, uint16_t *product_id* )

Register the specified vendor class device in the device table.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

### ◆ R_USB_HCDC_DeviceInfoGet()

fsp_err_t R_USB_HCDC_DeviceInfoGet ( usb_ctrl_t *const *p_api_ctrl*, usb_hcdc_device_info_t * *p_info*, uint8_t *device_address* )

Get the VID, PID and subclass code of the connected device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully completed. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*(1). Interrupt function.*
*(2). Callback function ( for RTOS ).*

## 5.2.6.37 USB HCDC_ECM (r_usb_hcdc_ecm)

Modules » Connectivity

### Functions

fsp_err_t R_USB_HCDC_ECM_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)

After USB, CDC are reset in software.Implements ether_api_t::open. More...

fsp_err_t R_USB_HCDC_ECM_Close (ether_ctrl_t *const p_ctrl)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More...

fsp_err_t R_USB_HCDC_ECM_BufferRelease (ether_ctrl_t *const p_ctrl)

Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease. More...

fsp_err_t R_USB_HCDC_ECM_RxBufferUpdate (ether_ctrl_t *const p_ctrl, void *const p_buffer)

Not support this function in HCDC-ECM. Implements ether_api_t::rxBufferUpdate. More...

| | |
|---|---|
| fsp_err_t | R_USB_HCDC_ECM_LinkProcess (ether_ctrl_t *const p_ctrl)<br>The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess. More... |
| fsp_err_t | R_USB_HCDC_ECM_WakeOnLANEnable (ether_ctrl_t *const p_ctrl)<br>Not support this function in HCDC-ECM. Implements ether_api_t::wakeOnLANEnable. More... |
| fsp_err_t | R_USB_HCDC_ECM_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const p_length_bytes)<br>Receive USB data. Receives data to the location specified by the pointer to the receive buffer. Implements ether_api_t::read. More... |
| fsp_err_t | R_USB_HCDC_ECM_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const length)<br>Transmit USB data. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. Implements ether_api_t::write. More... |
| fsp_err_t | R_USB_HCDC_ECM_TxStatusGet (ether_ctrl_t *const p_ctrl, void *const p_buffer_address)<br>Not support this function in HCDC-ECM. Implements ether_api_t::txStatusGet. More... |
| fsp_err_t | R_USB_HCDC_ECM_CallbackSet (ether_ctrl_t *const p_api_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory) |

### Detailed Description

This module provides a USB Host Communications Device Class (HCDC-ECM) driver. It implements the USB HCDC_ECM (r_usb_hcdc_ecm).

### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

### Detailed Description

# Overview

The r_usb_hcdc_ecm module, when used in combination with the r_usb_basic and r_usb_hcdc modules, operates as a USB Host Communications Device Class (HCDC-ECM) driver. The HCDC-ECM conforms to subclass ethernet control model of the USB Communications Device Class (CDC) specification and enables communication with a CDC peripheral device.

### Features

The r_usb_hcdc_ecm module has the following key features:

- Checks for connected devices
- Data transfer to and from a CDC-ECM peripheral device

# Configuration

### Build Time Configurations for r_usb_hcdc_ecm

The following build time configurations are defined in fsp_cfg/r_usb_hcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Target Peripheral Device Class ID | • CDC class supported device<br>• Vendor class device | CDC class supported device | Specify the device class ID of the CDC device to be connected. |

### Configurations for Connectivity > USB HCDC (r_usb_hcdc_ecm)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HCDC (r_usb_hcdc_ecm).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time a interrupt occurs. |
| Name | Name must be a valid C symbol | g_hcdc_ecm0 | Module name. |

*Note*

*Refer to the USB (r_usb_basic) module for hardware configuration options.*

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Basic Functions

The main functions of HCDC are the following:

- Verify connected devices
- Transfer data to and from the CDC-ECM peripheral device

### Class Request

This driver uses the following class request.

- SetEthernetPacketFilter

# Limitations

This driver has the following limitations:

### Dependencies

- This driver works on FreeRTOS and FreeRTOS Plus TCP.

### Requirements

- Please use a USB-LAN converter that supports the CDC-ECM class. If the USB-LAN converter does not support CDC-ECM class, this driver does not work properly.
- Please confirm with the manufacturer of using USB-LAN converter to see if it supports the CDC-ECM class.

### Constraints

This driver has the following constrains.

- Use of compound USB devices with CDC class support is not supported.
- This driver does not support Low-speed.
- This driver does not support simultaneous operation with the other device class.

# Operation Checking Converters

The following converters have been confirmed to work properly:

- UGREEN USB to Ethernet Adapter RJ45 (Manufacture: Ugreen Group Limited, Item model number: 50922)
- ELECOM USB to Ethernet Adapter EDC-GUA3V2-B (Manufacture: Elecom Co., Ltd, Item model number: EDC-GUA3V2-B)
- HP USB-C to RJ45 Adapter (Manufacture: Hewlett-Packard Company, Item model number: 7W66AA)

# Examples

### USB HCDC-ECM Example

The following is the porting layere example code for FreeRTOS Plus TCP

```
#define USR_TEST_DOMAIN_NAME "www.freertos.org"
#define USR_TEST_PING_IP "172.217.160.174"
#define USR_PING_COUNT 100
#define USR_APP_DELAY_TICKS 100
#define USR_PING_WAIT_MS 100
#define USR_PING_DATA_SIZE 8
#define MAC_ADDRESS_BYTES 6
#define SUCCESS 0
#define PRINT_UP_MSG_DISABLE 0x01
#define PRINT_DOWN_MSG_DISABLE 0x02
#define PRINT_NWK_USR_MSG_DISABLE 0x04
#define ETHERNET_LINK_DOWN 0x01
#define ETHERNET_LINK_UP 0x00
#define IP_LINK_DOWN 0x02
#define IP_LINK_UP 0x00
/* Domain for the DNS Host lookup is used in this Example Project.
 * The project can be built with different *domain_name to validate the DNS client
 */
char * domain_name = USR_TEST_DOMAIN_NAME;
/* IP address of the PC or any Device on the LAN/WAN where the Ping request is sent.
 * Note: Users needs to change this according to the LAN settings of your Test PC or
device
 * when running this project.
 */
#if (ipconfigUSE_DHCP != 0)
char ip_address_buffer[16] = {RESET_VALUE};
char * remote_ip_address = ip_address_buffer;
#else
char * remote_ip_address = "192.168.10.1";
#endif
#if (ipconfigSUPPORT_OUTGOING_PINGS == 1)
 #define PING_INTERVAL_TICKS 200
```

```
uint32_t usrPingCount = RESET_VALUE;
typedef struct st_ping_data
{
    uint32_t sent;                    // Ping Request
    uint32_t received;               // Ping Response
    uint32_t lost;                    // Ping failure
} ping_data_t;
ping_data_t ping_data = {RESET_VALUE, RESET_VALUE, RESET_VALUE};
#endif
BaseType_t xRet;
BaseType_t xEndPointCount = 0;
static NetworkInterface_t xInterfaces[1];
static NetworkEndPoint_t xEndPoints[4];
extern NetworkInterface_t * pxFSP_Eth_FillInterfaceDescriptor(BaseType_t xEMACIndex,
NetworkInterface_t * pxInterface);
#if (ipconfigUSE_DHCP != 0)
/* DHCP populates these IP address, Sub net mask and Gateway Address. So start with
this is zeroed out values
 * The MAC address is Test MAC address.
 */
static uint8_t ucMACAddress[6] = {RESET_VALUE};
static uint8_t ucIPAddress[4] = {RESET_VALUE};
static uint8_t ucNetMask[4] = {RESET_VALUE};
static uint8_t ucGatewayAddress[4] = {RESET_VALUE};
static uint8_t ucDNSServerAddress[4] = {RESET_VALUE};
#else
/* Static IP configuration, when DHCP mode is not used for the Example Project.
 * This needs to be populated by the user according to the Network Settings of your
LAN.
 * This sample address taken from the LAN where it is tested. This is different for
different LAN.
 * get the Address using the PC IPconfig details.
 */
static uint8_t ucMACAddress[6] = {RESET_VALUE};
```

```
static uint8_t ucIPAddress[4] = {192, 168, 10, 2};
static uint8_t ucNetMask[4] = {255, 255, 255, 0};
static uint8_t ucGatewayAddress[4] = {192, 168, 10, 0};
static uint8_t ucDNSServerAddress[4] = {RESET_VALUE};
#endif
#if (ipconfigUSE_DHCP != 0)
NetworkAddressingParameters_t xNetworkAddressing = {0, 0, 0, 0, 0};
NetworkAddressingParameters_t xNd                = {RESET_VALUE, RESET_VALUE,
RESET_VALUE, RESET_VALUE, RESET_VALUE};
#endif
#if (ipconfigUSE_DHCP != 0)
void updateDhcpResponseToUsr(void);
eDHCPCallbackAnswer_t xApplicationDHCPHook(eDHCPCallbackPhase_t eDHCPPhase, uint32_t
ulIPAddress);
eDHCPCallbackAnswer_t
xApplicationDHCPHook_Multi(eDHCPCallbackPhase_t      eDHCPPhase,
 struct xNetworkEndPoint * pxEndPoint,
                                                IP_Address_t            *
pxIPAddress);
#endif
static uint32_t usr_print_ability = RESET_VALUE;
uint32_t       dhcp_in_use       = RESET_VALUE;
const uint8_t * pucMACAddress = NULL;
uint32_t   isNetworkUp(void);
BaseType_t vSendPing(const char * pcIPAddress);
void pingSendFunc(void);
void print_ipconfig(void);
void print_pingResult(void);
void dnsQuerryFunc(char * domain_name);
BaseType_t vSendPing (const char * pcIPAddress)
{
    uint32_t ulIPAddress = RESET_VALUE;
 /*
  * The pcIPAddress parameter holds the destination IP address as a string in
```

```
  * decimal dot notation (for example, ?192.168.0.200?). Convert the string into
  * the required 32-bit format.
  */
    ulIPAddress = FreeRTOS_inet_addr(pcIPAddress);
 /*
  * Send a ping request containing 8 data bytes. Wait (in the Blocked state) a
  * maximum of 100ms for a network buffer into which the generated ping request
  * can be written and sent.
  */
 return FreeRTOS_SendPingRequest(ulIPAddress, USR_PING_DATA_SIZE, USR_PING_WAIT_MS /
portTICK_PERIOD_MS);
}
#if (ipconfigSUPPORT_OUTGOING_PINGS == 1)
void vApplicationPingReplyHook (ePingReplyStatus_t eStatus, uint16_t usIdentifier)
{
    (void) usIdentifier;
 switch (eStatus)
    {
 /* A valid ping reply has been received */
 case eSuccess:
            ping_data.received++;
 break;
 /* A reply was received but it was not valid. */
 case eInvalidData:
 default:
            ping_data.lost++;
 break;
    }
}
#endif
uint32_t isNetworkUp (void)
{
    BaseType_t networkUp = pdFALSE;
    uint32_t   status;
```

```
    networkUp = FreeRTOS_IsNetworkUp();
 if (pdTRUE == networkUp)
    {
       status = SUCCESS;
    }
 else
    {
       status = IP_LINK_DOWN;
    }
 return status;
}
void new_thread0_entry (void * pvParameters)
{
 fsp_pack_version_t version = {RESET_VALUE};
 FSP_PARAMETER_NOT_USED(pvParameters);
 /* version get API for FLEX pack information */
 R_FSP_VersionGet(&version);
 /* Example Project information printed on the RTT */
    APP_PRINT(BANNER_INFO,
              EP_VERSION,
              version.version_id_b.major,
              version.version_id_b.minor,
              version.version_id_b.patch);
 /* Prints the Ethernet Configuration prior to the IP Init*/
    APP_PRINT(ETH_PREINIT);
    print_ipconfig();
 /* Initialize the interface descriptor. */
    pxFSP_Eth_FillInterfaceDescriptor(0, &xInterfaces[0]);
 /* === End-point 0 === */
#if (ipconfigUSE_IPv4 != 0)
    FreeRTOS_FillEndPoint(&(xInterfaces[0]),
                          &(xEndPoints[xEndPointCount]),
                          ucIPAddress,
                          ucNetMask,
```

```
                        ucGatewayAddress,
                        ucDNSServerAddress,
                        ucMACAddress);
 #if (ipconfigUSE_DHCP != 0)
    {
 /* End-point 0 wants to use DHCPv4. */
       xEndPoints[xEndPointCount].bits.bWantDHCP = pdTRUE;
    }
 #endif  /* ( ipconfigUSE_DHCP != 0 ) */
    xEndPointCount += 1;
#endif  /* ( ipconfigUSE_IPv4 != 0 */
    FreeRTOS_IPInit_Multi();
    APP_PRINT(ETH_POSTINIT);
 while (true)
    {
 /* Check if Both the Ethernet Link and IP link are UP */
 if (SUCCESS == isNetworkUp())
     {
 /* usr_print_ability is added to avoid multiple UP messages or Down Messages
repeating*/
 if (!(PRINT_UP_MSG_DISABLE & usr_print_ability))
     {
                APP_PRINT("\r\nNetwork is Up");
                usr_print_ability |= PRINT_UP_MSG_DISABLE;
                pucMACAddress = FreeRTOS_GetMACAddress();
 if (NULL != pucMACAddress)
     {
                   memcpy(ucMACAddress, pucMACAddress, MAC_ADDRESS_BYTES);
     }
     }
 if (!(PRINT_NWK_USR_MSG_DISABLE & usr_print_ability))
     {
#if (ipconfigUSE_DHCP != 0)
 if (dhcp_in_use)
```

```
     {
 /* Display the New IP credentials obtained from the DHCP server */
                  updateDhcpResponseToUsr();
                  FreeRTOS_GetEndPointConfiguration(NULL,
                                                    &xNd.ulNetMask,
                                                    &xNd.ulGatewayAddress,
                                                    &xNd.ulDNSServerAddress,
                                                    &xEndPoints[0]);
 /* Updated IP credentials on to the RTT console */
     print_ipconfig();
 /*DNS lookup for the Domain name requested. This is Synchronous Activity */
                  dnsQuerryFunc(domain_name);
                  pingSendFunc();
                  print_pingResult();
                  usr_print_ability |= PRINT_NWK_USR_MSG_DISABLE;
     }
#else
 /* Updated IP credentials on to the RTT console */
     print_ipconfig();
               pingSendFunc();
               print_pingResult();
               usr_print_ability |= PRINT_NWK_USR_MSG_DISABLE;
#endif
     }
     }
 else
     {
 if (!(PRINT_DOWN_MSG_DISABLE & usr_print_ability))
     {
               APP_PRINT("\r\nNetwork is Down");
               usr_print_ability |= PRINT_DOWN_MSG_DISABLE;
     }
 else
     {
```

```
                APP_PRINT(".");
      }
      }
       vTaskDelay(USR_APP_DELAY_TICKS);
    }
}
#if (ipconfigUSE_DHCP != 0)
eDHCPCallbackAnswer_t xApplicationDHCPHook (eDHCPCallbackPhase_t eDHCPPhase, uint32_t
ulIPAddress)
{
    eDHCPCallbackAnswer_t eReturn = eDHCPContinue;
 /*
  * This hook is called in a couple of places during the DHCP process, as identified
by the eDHCPPhase parameter.
  */
 switch (eDHCPPhase)
    {
 case eDHCPPhasePreDiscover:
 /*
  * A DHCP discovery is about to be sent out. eDHCPContinue is returned to allow the
discovery to go out.
  * If eDHCPUseDefaults had been returned instead then the DHCP process would be
stopped and the statically
  * configured IP address would be used.
  * If eDHCPStopNoChanges had been returned instead then the DHCP process would be
stopped and whatever the
  * current network configuration was would continue to be used.
  */
 break;
 case eDHCPPhasePreRequest:
 /* An offer has been received from the DHCP server, and the offered IP address is
passed in the ulIPAddress
  * parameter.
  */
```

```
 /*
  * The sub-domains don?t match, so continue with the DHCP process so the offered IP
address is used.
  */
 /* Update the Structure, the DHCP state Machine is not updating this */
            xNetworkAddressing.ulDefaultIPAddress = ulIPAddress;
            dhcp_in_use = 1;
 break;
 default:
 /*
  * Cannot be reached, but set eReturn to prevent compiler warnings where compilers
are disposed to generating one.
  */
 break;
    }
 return eReturn;
}
eDHCPCallbackAnswer_t xApplicationDHCPHook_Multi
(eDHCPCallbackPhase_t      eDHCPPhase,
 struct xNetworkEndPoint * pxEndPoint,
                                                IP_Address_t            *
pxIPAddress)
{
    eDHCPCallbackAnswer_t eReturn = eDHCPContinue;
 FSP_PARAMETER_NOT_USED(pxEndPoint);
 /*
  * This hook is called in a couple of places during the DHCP process, as identified
by the eDHCPPhase parameter.
  */
 switch (eDHCPPhase)
    {
 case eDHCPPhasePreDiscover:
 /*
  * A DHCP discovery is about to be sent out. eDHCPContinue is returned to allow the
```

```
discovery to go out.
  * If eDHCPUseDefaults had been returned instead then the DHCP process would be
stopped and the statically
  * configured IP address would be used.
  * If eDHCPStopNoChanges had been returned instead then the DHCP process would be
stopped and whatever the
  * current network configuration was would continue to be used.
  */
 break;
 case eDHCPPhasePreRequest:
 /* An offer has been received from the DHCP server, and the offered IP address is
passed in the ulIPAddress
  * parameter.
  */
 /*
  * The sub-domains don?t match, so continue with the DHCP process so the offered IP
address is used.
  */
 /* Update the Structure, the DHCP state Machine is not updating this */
            xNetworkAddressing.ulDefaultIPAddress = pxIPAddress->ulIP_IPv4;
            dhcp_in_use = 1;
 break;
 default:
 /*
  * Cannot be reached, but set eReturn to prevent compiler warnings where compilers
are disposed to generating one.
  */
 break;
    }
 return eReturn;
}
#endif
void pingSendFunc (void)
{
```

```
#if (ipconfigSUPPORT_OUTGOING_PINGS == 1)
    BaseType_t status = pdFALSE;
    APP_PRINT("\r\n");
 while (usrPingCount < USR_PING_COUNT)
    {
 /* Send a ICMP Ping request to the requested IP address
  * USR_PING_COUNT (100) is used in this Example Project
  * For Continuous testing the count can be increased to bigger number
  */
       status = vSendPing((char *) remote_ip_address);
 if (status != pdFALSE)
      {
            ping_data.sent++;
            APP_PRINT("!");
      }
 else
      {
            ping_data.lost++;
            APP_PRINT(".");
      }
       usrPingCount++;
 /* Add some delay between Pings */
       vTaskDelay(PING_INTERVAL_TICKS);
    }
#endif
}
void print_pingResult (void)
{
#if (ipconfigSUPPORT_OUTGOING_PINGS == 1)
    APP_PRINT("\r\n \r\nPing Statistics for %s :\r\n", (char *) remote_ip_address);
    APP_PRINT("\r\nPackets: Sent = %02d, Received = %02d, Lost = %02d \r\n",
             ping_data.sent,
             ping_data.received,
             ping_data.lost);
```

```
#endif
}
void print_ipconfig (void)
{
#if (ipconfigUSE_DHCP != 0)
 if (dhcp_in_use)
    {
       ucNetMask[3] = (uint8_t) ((xNd.ulNetMask & 0xFF000000) >> 24);
       ucNetMask[2] = (uint8_t) ((xNd.ulNetMask & 0x00FF0000) >> 16);
       ucNetMask[1] = (uint8_t) ((xNd.ulNetMask & 0x0000FF00) >> 8);
       ucNetMask[0] = (uint8_t) (xNd.ulNetMask & 0x000000FF);
       ucGatewayAddress[3] = (uint8_t) ((xNd.ulGatewayAddress & 0xFF000000) >> 24);;
       ucGatewayAddress[2] = (uint8_t) ((xNd.ulGatewayAddress & 0x00FF0000) >> 16);
       ucGatewayAddress[1] = (uint8_t) ((xNd.ulGatewayAddress & 0x0000FF00) >> 8);
       ucGatewayAddress[0] = (uint8_t) (xNd.ulGatewayAddress & 0x000000FF);
       ucDNSServerAddress[3] = (uint8_t) ((xNd.ulDNSServerAddress & 0xFF000000) >>
24);
       ucDNSServerAddress[2] = (uint8_t) ((xNd.ulDNSServerAddress & 0x00FF0000) >>
16);
       ucDNSServerAddress[1] = (uint8_t) ((xNd.ulDNSServerAddress & 0x0000FF00) >>
8);
       ucDNSServerAddress[0] = (uint8_t) (xNd.ulDNSServerAddress & 0x000000FF);
       ucIPAddress[3] = (uint8_t) ((xNd.ulDefaultIPAddress & 0xFF000000) >> 24);
       ucIPAddress[2] = (uint8_t) ((xNd.ulDefaultIPAddress & 0x00FF0000) >> 16);
       ucIPAddress[1] = (uint8_t) ((xNd.ulDefaultIPAddress & 0x0000FF00) >> 8);
       ucIPAddress[0] = (uint8_t) (xNd.ulDefaultIPAddress & 0x000000FF);
    }
#endif
    APP_PRINT("\r\nEthernet adapter for Renesas "KIT_NAME ":\r\n")
    APP_PRINT("\tDescription . . . . . . . . . . . : Renesas "KIT_NAME "
Ethernet\r\n");
    APP_PRINT("\tPhysical Address. . . . . . . . . :
%02x-%02x-%02x-%02x-%02x-%02x\r\n",
             ucMACAddress[0],
```

```
            ucMACAddress[1],
            ucMACAddress[2],
            ucMACAddress[3],
            ucMACAddress[4],
            ucMACAddress[5]);
    APP_PRINT("\tDHCP Enabled. . . . . . . . . . . : %s\r\n", dhcp_in_use ? "Yes" :
"No")
    APP_PRINT("\tIPv4 Address. . . . . . . . . . . : %d.%d.%d.%d\r\n",
            ucIPAddress[0],
            ucIPAddress[1],
            ucIPAddress[2],
            ucIPAddress[3]);
    APP_PRINT("\tSubnet Mask . . . . . . . . . . . : %d.%d.%d.%d\r\n",
            ucNetMask[0],
            ucNetMask[1],
            ucNetMask[2],
            ucNetMask[3]);
    APP_PRINT("\tDefault Gateway . . . . . . . . . : %d.%d.%d.%d\r\n",
            ucGatewayAddress[0],
            ucGatewayAddress[1],
            ucGatewayAddress[2],
            ucGatewayAddress[3]);
    APP_PRINT("\tDNS Servers . . . . . . . . . . . : %d.%d.%d.%d\r\n",
            ucDNSServerAddress[0],
            ucDNSServerAddress[1],
            ucDNSServerAddress[2],
            ucDNSServerAddress[3]);
}
void dnsQuerryFunc (char * domain)
{
    uint32_t ulIPAddress = RESET_VALUE;
    int8_t   cBuffer[16] = {RESET_VALUE};
 /* Lookup the IP address of the FreeRTOS.org website. */
    ulIPAddress = FreeRTOS_gethostbyname((char *) domain);
```

```
 if (ulIPAddress != 0)
    {
 /* Convert the IP address to a string. */
       FreeRTOS_inet_ntoa(ulIPAddress, (char *) cBuffer);
 /* Print out the IP address obtained from the DNS lookup. */
       APP_PRINT("\r\nDNS Lookup for \"www.freertos.org\" is      : %s \r\n",
cBuffer);
 /* Overwrites the user-specified Ping request destination with the IP address
obtained from the domain.*/
       memcpy((void *) remote_ip_address, (const void *) cBuffer, strlen((const char
*) cBuffer) + 1);
    }
 else
    {
       APP_PRINT("\r\nDNS Lookup failed for \"www.freertos.org\" \r\n");
       remote_ip_address = USR_TEST_PING_IP;
    }
}
#if (ipconfigUSE_DHCP != 0)
void updateDhcpResponseToUsr (void)
{
 if (dhcp_in_use)
    {
       memcpy(&xNd, &xNetworkAddressing, sizeof(xNd));
    }
}
#endif
#if (ipconfigDHCP_REGISTER_HOSTNAME == 1)
const char * pcApplicationHostnameHook (void)
{
 return KIT_NAME;
}
#endif
```

## Data Structures

| | |
|---|---|
| struct | usb_hcdc_ecm_instance_ctrl_t |

## Data Structure Documentation

### ◆ usb_hcdc_ecm_instance_ctrl_t

| struct usb_hcdc_ecm_instance_ctrl_t | |
|---|---|
| USB HCDC ECM control block. DO NOT INITIALIZE. Initialization occurs when ether_api_t::open is called. | |
| **Public Member Functions** | |
| uint8_t read_buffer [HCDC_ECM_READ_DATA_LEN] | __attribute__ ((__aligned__(4))) |
| | Buffer area for read data. |
| | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| uint32_t | receive_size |
| | Received data size. |
| | |
| uint32_t | receive_status |
| | Status of receive result. |
| | |

## Function Documentation

### ◆ R_USB_HCDC_ECM_Open()

fsp_err_t R_USB_HCDC_ECM_Open ( ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg* )

After USB, CDC are reset in software.Implements ether_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_USB_NOT_OPEN | USB module is not open.. |
| FSP_ERR_ALREADY_OPEN | RTOS internal error. |

### ◆ R_USB_HCDC_ECM_Close()

fsp_err_t R_USB_HCDC_ECM_Close ( ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |

### ◆ R_USB_HCDC_ECM_BufferRelease()

fsp_err_t R_USB_HCDC_ECM_BufferRelease ( ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported. |

### ◆ R_USB_HCDC_ECM_RxBufferUpdate()

fsp_err_t R_USB_HCDC_ECM_RxBufferUpdate ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer* )

Not support this function in HCDC-ECM. Implements ether_api_t::rxBufferUpdate.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported. |

### ◆ R_USB_HCDC_ECM_LinkProcess()

fsp_err_t R_USB_HCDC_ECM_LinkProcess ( ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Link is up. |
| FSP_ERR_USB_FAILED | Link is not up. |

### ◆ R_USB_HCDC_ECM_WakeOnLANEnable()

fsp_err_t R_USB_HCDC_ECM_WakeOnLANEnable ( ether_ctrl_t *const *p_ctrl*)

Not support this function in HCDC-ECM. Implements ether_api_t::wakeOnLANEnable.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported. |

### ◆ R_USB_HCDC_ECM_Read()

fsp_err_t R_USB_HCDC_ECM_Read ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *p_length_bytes* )

Receive USB data. Receives data to the location specified by the pointer to the receive buffer. Implements ether_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_STATE | USB data read failure. |

### ◆ R_USB_HCDC_ECM_Write()

fsp_err_t R_USB_HCDC_ECM_Write ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *length* )

Transmit USB data. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. Implements ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_STATE | USB data write failure. |
| FSP_ERR_INVALID_ARGUMENT | Value of the send frame size is out of range. |

### ◆ R_USB_HCDC_ECM_TxStatusGet()

fsp_err_t R_USB_HCDC_ECM_TxStatusGet ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer_address* )

Not support this function in HCDC-ECM. Implements ether_api_t::txStatusGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is not supported. |

### ◆ R_USB_HCDC_ECM_CallbackSet()

fsp_err_t R_USB_HCDC_ECM_CallbackSet ( ether_ctrl_t *const *p_api_ctrl*, void(*)(ether_callback_args_t *) *p_callback*, void *const *p_context*, ether_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements ether_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.6.38 USB HHID (r_usb_hhid)

Modules » Connectivity

### Functions

fsp_err_t R_USB_HHID_TypeGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.) More...

fsp_err_t R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT). More...

### Detailed Description

This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface.

# Overview

The r_usb_hhid module combines with the r_usb_basic module to provide a USB Host Human Interface Device Class (HHID) driver. The HHID driver conforms to the USB Human Interface Device class specifications and implements communication with a HID device.

### Features

The r_usb_hhid module has the following key features:

- Data communication with a connected HID device (USB mouse, keyboard etc.)
- Issuing of HID class requests to a connected HID device
- Supports Interrupt OUT transfer

# Configuration

### Build Time Configurations for r_usb_hhid

The following build time configurations are defined in fsp_cfg/r_usb_hhid_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| USB Hub Multi HID | • Disabled<br>• Enabled | Disabled | Enable or disable HUB support for HID-HID. |

### Configurations for Connectivity > USB HHID (r_usb_hhid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HHID (r_usb_hhid). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hhid0 | Module name. |

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

*Note*

*This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB peripherals.*

### Class Requests

The class requests supported by this driver are shown below:

| Request | Code | Description |
|---|---|---|
| USB_GET_REPORT | 0x01 | Receives a report from the HID device. |
| USB_SET_REPORT | 0x09 | Sends a report to the HID device. |
| USB_GET_IDLE | 0x02 | Receives a duration (time) from the HID device. |
| USB_SET_IDLE | 0x0A | Sends a duration (time) to the HID device. |
| USB_GET_PROTOCOL | 0x03 | Reads a protocol from the HID device. |
| USB_SET_PROTOCOL | 0x0B | Sends a protocol to the HID device. |
| USB_GET_REPORT_DESCRIPTOR | 0x06 | Requests a report descriptor. |
| USB_GET_HID_DESCRIPTOR | 0x06 | Requests a HID descriptor. |

### Data Format

The boot protocol data format of data received from the keyboard or mouse through interrupt-IN transfers is shown below:

| offset | Keyboard (8 Bytes) | Mouse (3 Bytes) |
|---|---|---|
| 0 (Top Byte) | Modifier keys | b0 : Button 1<br>b1 : Button 2<br>b2 : Button 3<br>b3-b7 : Reserved |
| +1 | Reserved | X displacement |
| +2 | Keycode 1 | Y displacement |
| +3 | Keycode 2 | - |
| +4 | Keycode 3 | - |
| +5 | Keycode 4 | - |
| +6 | Keycode 5 | - |
| +7 | Keycode 6 | - |

### Limitations

- The HID driver does not analyze the report descriptor. This driver determines the report format from the interface protocol.
- This driver does not support DMA transfers.
- This driver does not support High-speed.

- The transfer rates of Full-speed and Low-speed are the same when the max packet sizes of Full-speed and Low-speed are the same.
- This driver does not support simultaneous operation with the other device class.

# Examples

### USB HHID Example

The main functions of the application are as follows:

1. Performs enumeration and initialization of HID devices.
2. Transfers data to and from a connected HID device (mouse or keyboard). Data received from the device is read and discarded.
3. When an RTOS is used, the USB driver calls the callback (usb_apl_callback) in order to pass events to the main loop through a queue.

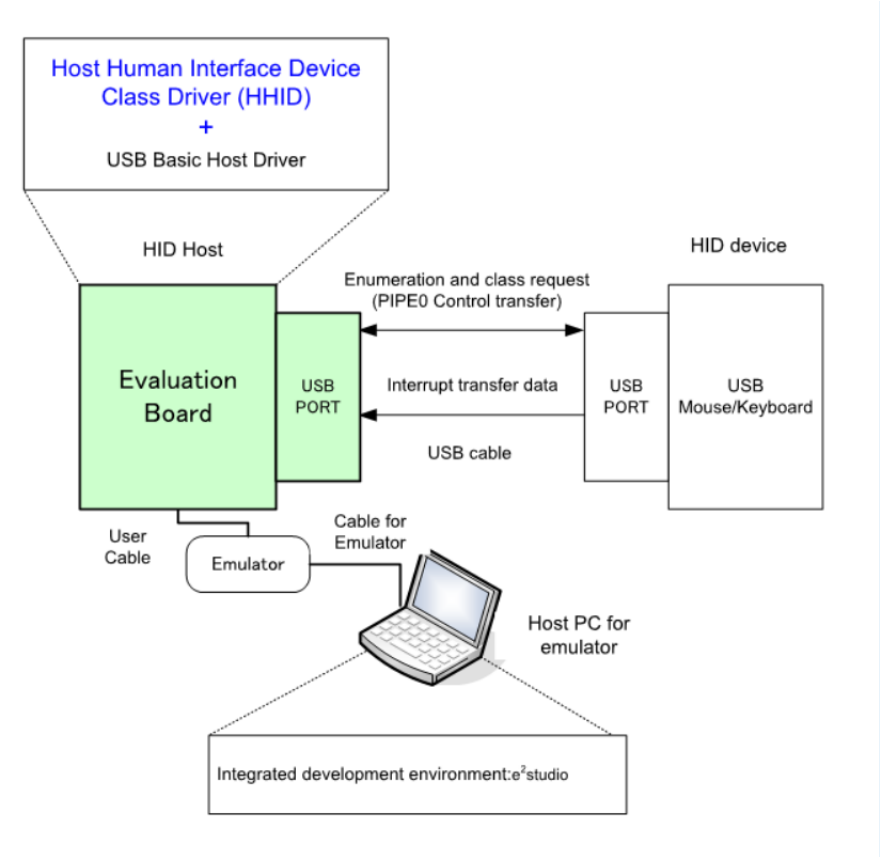



Figure 236: Example Operating Environment

### Application Processing (for RTOS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback

function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_CONFIGURED, APL sends the class request (SET_PROTOCOL) to the HID device.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_REQUEST_COMPLETE, APL performs a data reception request to receive data transmitted from the HID device by calling the R_USB_Read function.
5. The above processing is repeated.

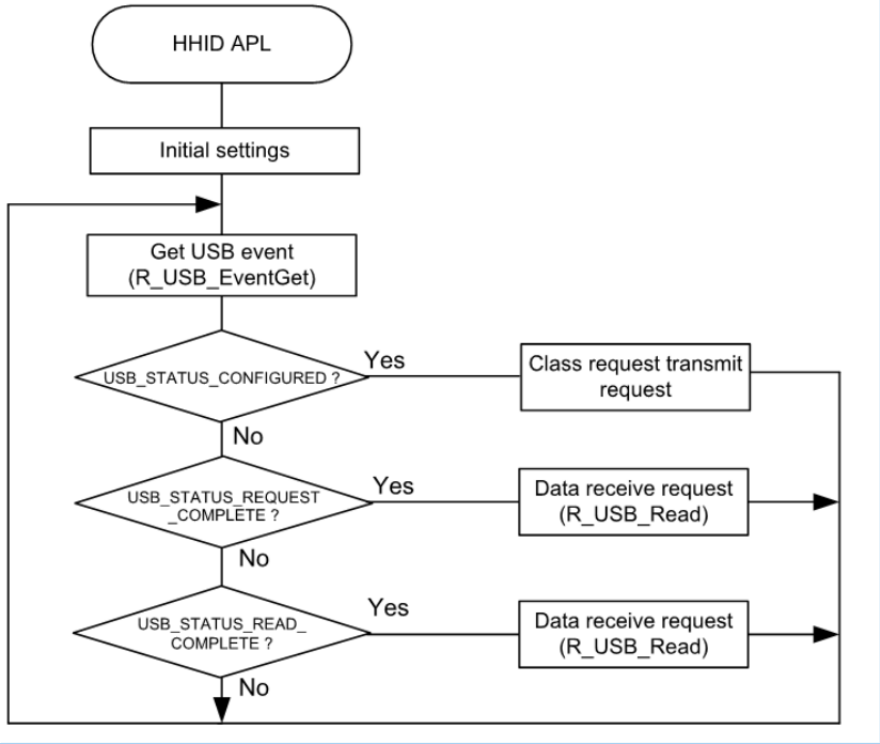



Figure 237: Main Loop (Normal mode)

### Application Processing (for Non-OS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after an HID device attaches to the USB host and enumeration completes, USB_STATUS_CONFIGURED is set as the return value. When the APL confirms USB_STATUS_CONFIGURED, it calls the R_USB_Write function to request transmission of data to the HID device.
2. When the R_USB_GetEvent function is called after sending of class request SET_PROTOCOL to the HID device has completed, USB_STATUS_REQUEST_COMPLETE is set as the return value. When the APL confirms USB_STATUS_REQUEST_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
3. When the R_USB_GetEvent function is called after reception of data from the HID device has completed, USB_STATUS_READ_COMPLETE is set as the return value. When the APL confirms USB_STATUS_READ_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
4. The processing in step 3, above, is repeated.

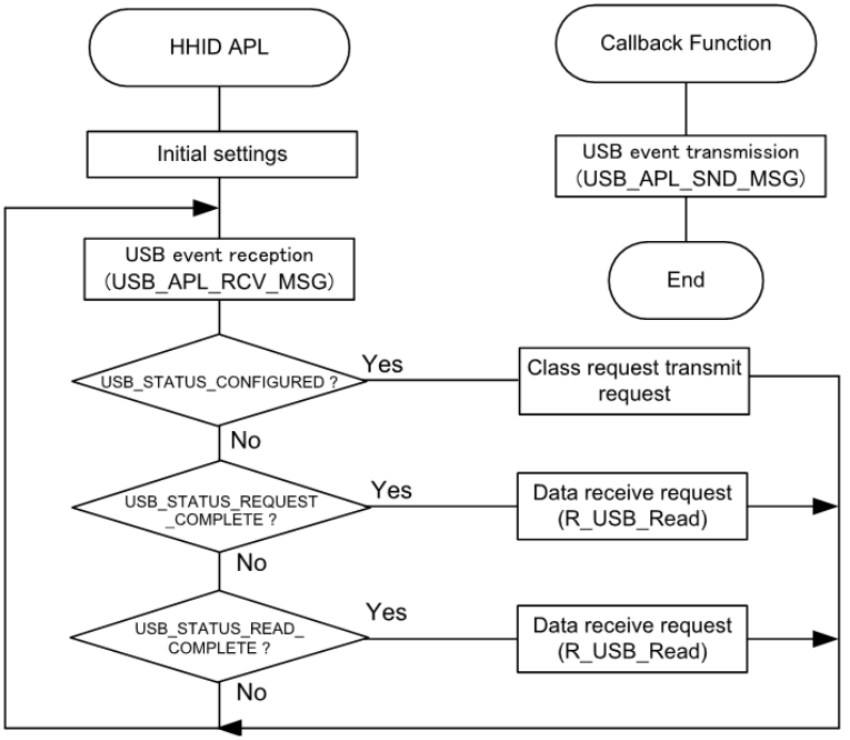



Figure 238: Main Loop (Normal mode)

```
/***********************************************************************************
 * Macro definitions
 ***********************************************************************************/
#define SET_PROTOCOL (USB_HID_SET_PROTOCOL | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define BOOT_PROTOCOL (0)
#define USB_FS_DEVICE_ADDRESS_1 (1)
/***********************************************************************************
 * Private global variables and functions
 ***********************************************************************************/
static const usb_hhid_api_t g_hhid_on_usb =
{
    .typeGet          = R_USB_HHID_TypeGet,
    .maxPacketSizeGet = R_USB_HHID_MaxPacketSizeGet,
};
#if (BSP_CFG_RTOS == 2)
/***********************************************************************************
 * Function Name : usb_apl_callback
```

```
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ***********************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/***********************************************************************
 * Function Name : usb_hhid_example
 * Description : Host HID application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************/
void usb_hhid_example (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif  /* (BSP_CFG_RTOS == 2) */
 usb_status_t    event;
    usb_event_info_t event_info;
    uint16_t         offset = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
```

```
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event); /* Get event code */
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                g_hhid_on_usb.typeGet(&g_basic0_ctrl, &g_hid_type,
USB_FS_DEVICE_ADDRESS_1);
                g_hhid_on_usb.maxPacketSizeGet(&g_basic0_ctrl, &g_mxps, USB_HID_IN,
USB_FS_DEVICE_ADDRESS_1);
 /* Send the HID request (SetProtocol) to HID device */
                set_protocol(&g_basic0_ctrl, BOOT_PROTOCOL, USB_FS_DEVICE_ADDRESS_1);
 break;
           }
 case USB_STATUS_READ_COMPLETE:
      {
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
      {
                  offset = hid_memcpy(g_store_buf, g_buf, offset, g_mxps);
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
             }
 break;
           }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_HID_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
             }
```

```
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_hhid_example() */
/******************************************************************************
 * Function Name : set_protocol
 * Description : Sending SetProtocol request to HID device
 * Arguments : usb_ctrl_t *p_ctrl : Pointer to usb_instance_ctrl_t structure.
 * : uint8_t ptorocol: Protocol Type
 * : uint8_t device_address: Device address that sends this request
 * Return value : none
 ******************************************************************************/
static void set_protocol (usb_instance_ctrl_t * p_ctrl, uint8_t protocol, uint8_t
device_address)
{
    usb_setup_t setup;
    setup.request_type   =
SET_PROTOCOL;                                                    /*
bRequestCode:SET_PROTOCOL, bmRequestType */
    setup.request_value =
protocol;                                                        /* wValue:
Protocol Type */
    setup.request_index =
0x0000;                                                          /*
wIndex:Interface */
    setup.request_length =
0x0000;                                                          /* wLength:Zero
*/
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_setup_data,
```

```
device_address); /* Request Control transfer */
} /* End of function set_protocol() */
/******************************************************************************
 * Function Name : hid_memcpy
 * Description : Copy received hhid data to the application buffer
 * Arguments : uint8_t *p_dest : Pointer to application buffer
 * : uint8_t *p_src : Pointer to received buffer
 * : uint16_t offset : Application buffer offset
 * : uint16_t size : Size of receiced hhid data
 * Return value : uint16_t offset + i: Offset
 ******************************************************************************/
static uint16_t hid_memcpy (uint8_t * p_dest, uint8_t * p_src, uint16_t offset,
uint16_t size)
{
    uint16_t i;
 for (i = 0; i < size; i++)
    {
 if (BUFSIZE == (offset + i))
     {
           offset = 0;
      }
      *(p_dest + offset + i) = *(p_src + i);
    }
 return (uint16_t) (offset + i);
} /* End of function hid_memcpy() */
```

## Function Documentation

### ◆ R_USB_HHID_TypeGet()

fsp_err_t R_USB_HHID_TypeGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_type*, uint8_t *device_address* )

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HHID_MaxPacketSizeGet()

fsp_err_t R_USB_HHID_MaxPacketSizeGet ( usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_size*, uint8_t *direction*, uint8_t *device_address* )

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

## 5.2.6.39 USB HMSC (r_usb_hmsc)

Modules » Connectivity

**Functions**

| | |
|---|---|
| fsp_err_t | R_USB_HMSC_StorageCommand (usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)<br>Processing for MassStorage(ATAPI) command. More... |
| fsp_err_t | R_USB_HMSC_DriveNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination) |

| | |
|---|---|
| | Get number of Storage drive. More... |
| fsp_err_t | R_USB_HMSC_StorageReadSector (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)<br>Read sector information. More... |
| fsp_err_t | R_USB_HMSC_StorageWriteSector (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)<br>Write sector information. More... |
| fsp_err_t | R_USB_HMSC_SemaphoreGet (void)<br>Get a semaphore. (RTOS only) More... |
| fsp_err_t | R_USB_HMSC_SemaphoreRelease (void)<br>Release a semaphore. (RTOS only) More... |

### Detailed Description

This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface.

# Overview

The r_usb_hmsc module, when used in combination with the r_usb_basic module, operates as a USB Host Mass Storage Class (HMSC) driver. It is built on the USB Mass Storage Class Bulk-Only Transport (BOT) protocol. It is possible to communicate with BOT-compatible USB storage devices by combining this module with a file system and storage device driver.

*Note*

*This module should be used in combination with the FreeRTOS+FAT File System.*

### Features

The r_usb_hmsc module has the following key features:

- Checking of connected USB storage devices to determine whether or not operation is supported
- Storage command communication using the BOT protocol
- Support for SFF-8070i (ATAPI) USB mass storage subclass
- Sharing of a single pipe for IN/OUT directions or multiple devices
- Supports up to 4 connected USB storage devices

### Known Issues

- Users can not use TrustZone features that use Non-secure Callable.

### Class Requests

The class requests supported by this driver are shown below.

| Request | Description |
|---|---|
| GetMaxLun | Gets the maximum number of units that are supported. |
| MassStorageReset | Cancels a protocol error. |

### Storage Commands

This driver supports the following storage commands:

- TEST_UNIT_READY
- MODE_SELECT10
- MODE_SENSE10
- PREVENT_ALLOW
- READ_FORMAT_CAPACITY
- READ10
- WRITE10

# Configuration

### Build Time Configurations for r_usb_hmsc

The following build time configurations are defined in fsp_cfg/r_usb_hmsc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| USB Hub Multi MSC | • Disabled<br>• Enabled | Disabled | Enable or disable HUB support for MSC-MSC. |

Refer to the USB (r_usb_basic) module.

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

Warning

Due to the wide variety of USB mass storage device implementations, this driver is not guaranteed to work with all devices. When implementing the driver it is important to verify correct operation with the mass storage devices that the end user is expected to use.

### Multi Port

This driver supports simultaneous operation with Peripheral Communication Device Class(PCDC). If the user are using MCU that supports 2 USB modules, such as RA6M3, the user can run HMSC on one USB module and PCDC on the other. This driver does not support simultaneous operation using device classes other than PCDC.

### For Bare Metal

1. To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.
2. In RA configurator, enter the appropriate values in the Main stack size and Heap size fields. The figure below is an example of the RA6M3-EK board.

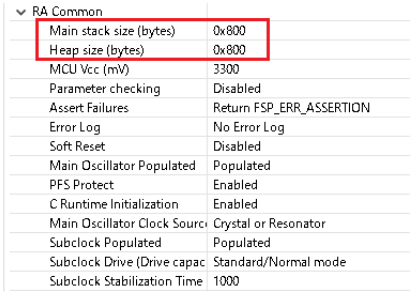



Figure 239: BSP Setting

1. In the Bare Metal version, specify "NULL" in the Callback item.

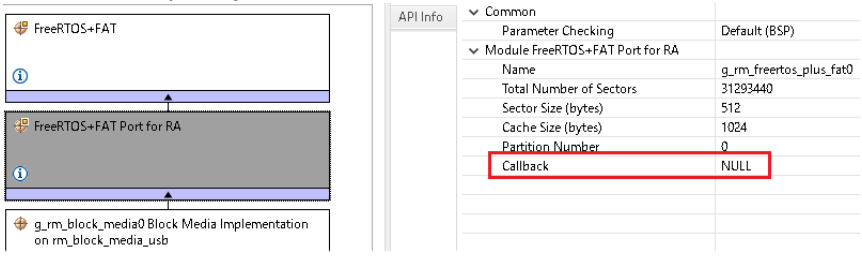



Figure 240: For Bare Metal Setting

### Limitations

1. Some MSC devices may be unable to connect because they are not recognized as storage devices.
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. A maximum of 4 USB storage devices can be connected.
4. Only USB storage devices with a sector size of 512 bytes can be connected.
5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
6. The continuous transfer mode cannot be used when using DMA.
7. This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
8. This driver does not support Low-speed.

# Examples

## USB HMSC Example

### Example Operating Environment

The following shows an example operating environment for the HMSC.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

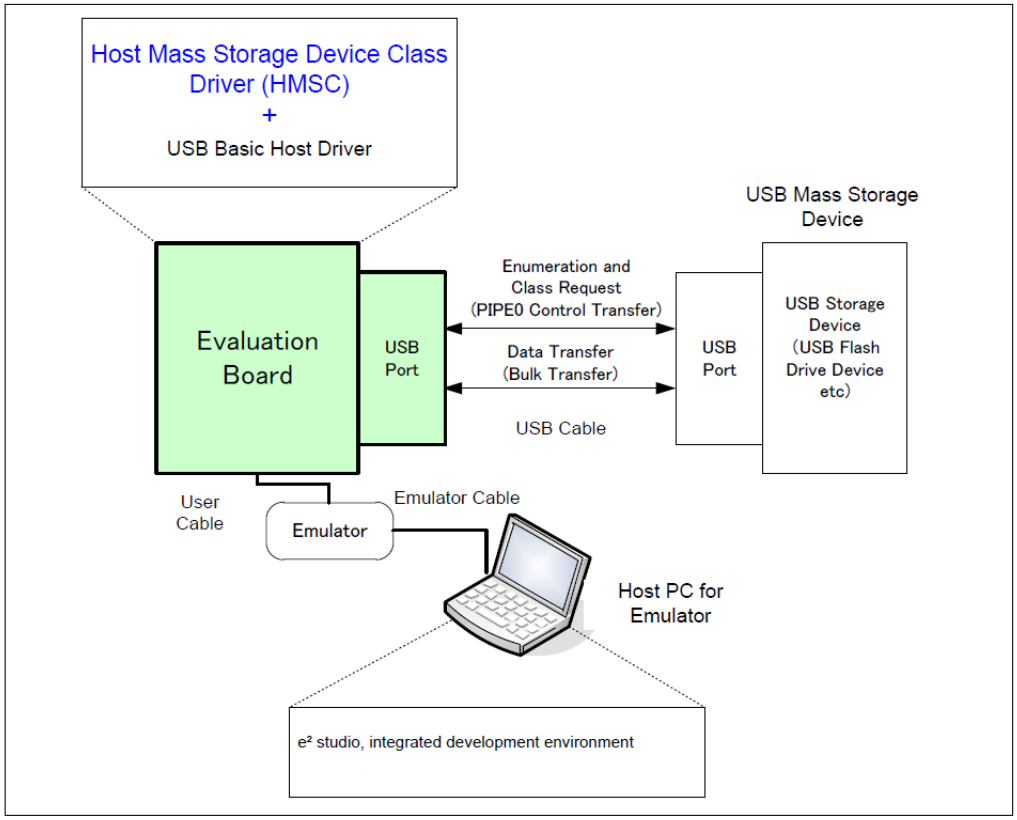



Figure 241: Example Operating Environment

### Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on MSC devices.
2. After the above processsing finisihes, the application writes the file to the MSC device once.
3. After writing the above file, the APL repeatedly reads the file. It continues to read the file repeatedly until the switch is pressed again.

### Application Processing (for RTOS)

This application has two tasks. An overview of the processing in these two tasks is provided below.

#### usb_apl_task

1. After start up, MCU pin setting, USB controller initialization, and application program initialization are performed.

2. The MSC device is attached to the kit. When enumeration and drive recognition processing have completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task is notified of the USB completion event using the FreeRTOS functionality.
3. In the application task, information regarding the USB completion event about which notification was received from the callback function is retrieved using the real-time OS functionality.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED then, based on the USB completion event, the MSC device is mounted and the file is written to the MSC device.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_DETACH, the application initializes the variables for state management.

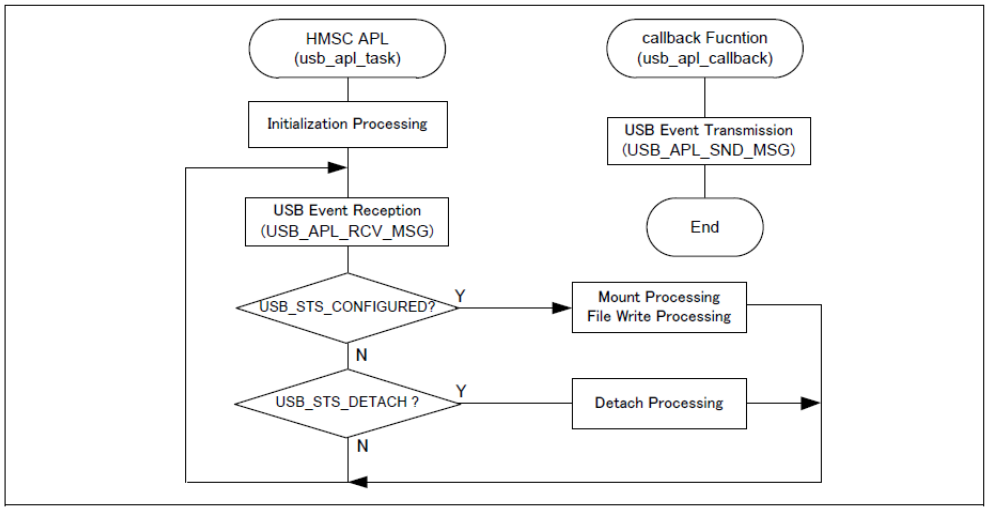



Figure 242: usb_apl_task

### file_read_task

Of the application tasks usb_apl_task and file_read_task, file_read_task is processed while usb_apl_task is in the wait state. This task performs file read processing on the file that was written to the MSC device.

**Example Code**

```
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
typedef enum
{
    STATE_ATTACH, STATE_DATA_READY, STATE_DATA_WRITE, STATE_FILE_READ, STATE_DETACH,
STATE_ERROR,
} state_t;
```

```
extern rm_freertos_plus_fat_instance_ctrl_t g_rm_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_rm_freertos_plus_fat0_cfg;
// @@extern const rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat0_disk_cfg;
uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
static uint16_t g_state = STATE_DETACH;
void usb_hmsc_baremetal_example (void)
{
    uint16_t i;
    uint16_t k;
 fsp_err_t err;
    FF_FILE * pxSourceFile;
    FF_Disk_t disk;
 rm_freertos_plus_fat_device_t device;
 usb_status_t    event;
    usb_event_info_t event_info;
    FF_Error_t      ff_err;
 size_t          size_return;
 int             close_err;
 rm_block_media_usb_instance_ctrl_t * p_instance_ctrl;
 for (i = 0; i < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; i++)
    {
       g_file_data[i] = (uint8_t) i;
    }
 /* Open media driver.*/
 RM_FREERTOS_PLUS_FAT_Open(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_cfg);
/* When using USB media, enable RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB macro. */
 #ifdef RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
     {
```

```
 case USB_STATUS_CONFIGURED:
      {
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
                p_instance_ctrl                = event_info.p_context;
                p_instance_ctrl->device_address = event_info.device_address;
 RM_FREERTOS_PLUS_FAT_MediaInit(&g_rm_freertos_plus_fat0_ctrl, &device);
 /* Initialize one disk for each partition used in the application. */
 RM_FREERTOS_PLUS_FAT_DiskInit(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_disk_cfg, &disk);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
                FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 /* Add the disk to the file system. */
                FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
                pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
 /* Write file data. */
                ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
 /* Close the file. */
                ff_fclose(pxSourceFile);
                g_state = STATE_FILE_READ;
 break;
      }
 case USB_STATUS_DETACH:
      {
                g_state = STATE_DETACH;
 RM_FREERTOS_PLUS_FAT_DiskDeinit(&g_rm_freertos_plus_fat0_ctrl, &disk);
 break;
      }
 default:
      {
 break;
```

```
     }
     }
 if (STATE_FILE_READ == g_state)
     {
           pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "r");
 for (k = 0; k < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; k++)
     {
               g_read_buffer[k] = (uint8_t) 0;
     }
 /* Read file data. */
           size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1,
pxSourceFile);
 /* Close the file. */
           close_err = ff_fclose(pxSourceFile);
     }
   }
 #endif
}
```

### Function Documentation

#### ◆ R_USB_HMSC_StorageCommand()

| fsp_err_t R_USB_HMSC_StorageCommand ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *buf*, uint8_t *command*, uint8_t *destination* ) |
|---|
| Processing for MassStorage(ATAPI) command. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HMSC_DriveNumberGet()

fsp_err_t R_USB_HMSC_DriveNumberGet ( usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_drive*, uint8_t *destination* )

Get number of Storage drive.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

### ◆ R_USB_HMSC_StorageReadSector()

fsp_err_t R_USB_HMSC_StorageReadSector ( uint16_t *drive_number*, uint8_t *const *buff*, uint32_t *sector_number*, uint16_t *sector_count* )

Read sector information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument buff must be 4-byte aligned.*

### ◆ R_USB_HMSC_StorageWriteSector()

fsp_err_t R_USB_HMSC_StorageWriteSector ( uint16_t *drive_number*, uint8_t const *const *buff*, uint32_t *sector_number*, uint16_t *sector_count* )

Write sector information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |
| FSP_ERR_ASSERTION | Parameter Null pointer error. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

***Note***

*The address specified in the argument buff must be 4-byte aligned.*

### ◆ R_USB_HMSC_SemaphoreGet()

fsp_err_t R_USB_HMSC_SemaphoreGet ( void )

Get a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |

### ◆ R_USB_HMSC_SemaphoreRelease()

fsp_err_t R_USB_HMSC_SemaphoreRelease ( void )

Release a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_USB_FAILED | The function could not be completed successfully. |

### 5.2.6.40 USB Host Vendor class (r_usb_hvnd)

Modules » Connectivity

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

# Overview

USB Host Vendor class works by combining r_usb_basic module.

# How to Configuration

The following shows FSP configuration procedure for USB Host Vendor class.

- Select [New Stack]->[Middleware]->[USB]->[USB Host Vendor class driver on r_usb_hvnd].

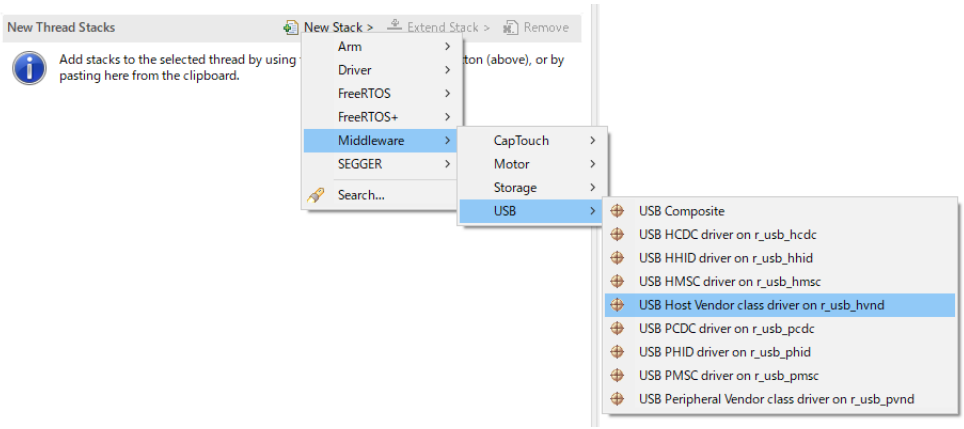



Figure 243: Select USB Host Vendor Class

- The following is displayed when selecting [USB Host Vendor class driver on r_usb_hvnd]. The user does not specify USB pipe number in Vendor class.

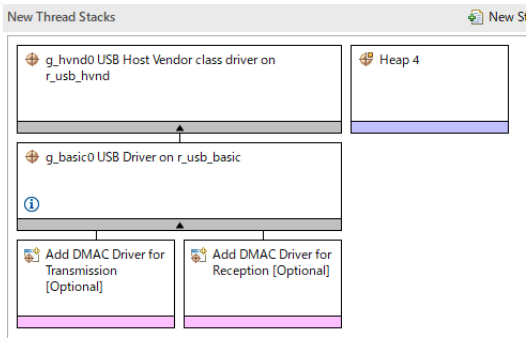



Figure 244: USB Host Vendor Class Stack

# API

Use the following APIs in Host Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()
  3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_HostControlTransfer()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

# USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.

# Examples

This application program processes the following after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion
2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/***********************************************************************
**
 * Macro definitions
 ***********************************************************************
**/
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
```

```
#define SET_VENDOR_NO_DATA (USB_SET_VENDOR_NO_DATA | USB_HOST_TO_DEV |
USB_VENDOR | USB_INTERFACE)
#define SET_VENDOR (USB_SET_VENDOR | USB_HOST_TO_DEV | USB_VENDOR |
USB_INTERFACE)
#define GET_VENDOR (USB_GET_VENDOR | USB_DEV_TO_HOST | USB_VENDOR |
USB_INTERFACE)
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
**
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************
**/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
**
 * Function Name : usb_hvnd_example
 * Description : Peripheral Vendor Class application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
void usb_hvnd_example (void)
```

```
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type = 0;
    uint8_t    pipe      = 0;
    uint8_t    is_zlp[2] = {0, 0};
    uint16_t   used_pipe = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
     {
 case USB_STATUS_CONFIGURED:
     {
               buffer_init();
               is_zlp[0] = 0;
               is_zlp[1] = 0;
 /* Get USB Pipe Information */
               g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
```

```
ADDRESS1);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if (0 != (used_pipe & (1 << pipe)))
      {
                        g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                            buf_type      = BUF_BULK;
                            bulk_out_pipe = pipe;
                        }
 else
      {
                            buf_type    = BUF_INT;
                            int_out_pipe = pipe;
                        }
                    }
 else
      {
 /* In Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                            buf_type    = BUF_BULK;
                            bulk_in_pipe = pipe;
                        }
 else
      {
                            buf_type    = BUF_INT;
                            int_in_pipe = pipe;
                        }
```

```
                        }
                    }
                }
 /* Send Vendor Class Request */
                class_request_set_vendor_no_data(&g_basic0_ctrl,
event_info.device_address);
 break;
            }
 case USB_STATUS_READ_COMPLETE:
     {
 if (FSP_ERR_USB_FAILED != event_info.status)
     {
 if (bulk_in_pipe == event_info.pipe)
     {
                        buf_type = BUF_BULK;
                        pipe    = bulk_out_pipe;
                    }
 else if (int_in_pipe == event_info.pipe)
     {
                        buf_type = BUF_INT;
                        pipe    = int_out_pipe;
                    }
 else
     {
 while (1)
     {
                            ;
                        }
                    }
                    buffer_check(buf_type, event_info.data_size);
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
                }
 break;
```

```
            }
 case USB_STATUS_WRITE_COMPLETE:
      {
 if (FSP_SUCCESS == event_info.status)
      {
 if (bulk_out_pipe == event_info.pipe)
      {
                        buf_type = BUF_BULK;
 if (1 == is_zlp[buf_type])
      {
                            pipe = bulk_in_pipe;
                        }
                    }
 else if (int_out_pipe == event_info.pipe)
      {
                        buf_type = BUF_INT;
 if (1 == is_zlp[buf_type])
      {
                            pipe = int_in_pipe;
                        }
                    }
 else
      {
 /* Nothing */
                    }
 if (1 == is_zlp[buf_type])
      {
                        is_zlp[buf_type] = 0;
                        buffer_clear(buf_type);
                        g_usb_on_usb.pipeRead(&g_basic0_ctrl,
&g_buf[buf_type][0], BUF_SIZE, pipe);
                    }
 else
      {
```

```
                        is_zlp[buf_type] = 1;
                        g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                    }
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
      {
                    class_request_set_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                    class_request_get_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                  buffer_init();
 /* Bulk Out Transfer */
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[BUF_BULK][0], (BUF_SIZE - USB_APL_MXPS),
                                           bulk_out_pipe);
 /* Interrupt Out Transfer */
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl, &g_buf[BUF_INT][0],
(BUF_SIZE - USB_APL_MXPS), int_out_pipe);
                }
 else
      {
 /* Unsupported request */
```

```
                }
 break;
            }
 case USB_STATUS_DETACH:
     {
 break;
            }
 default:
     {
 break;
            }
      }
   }
} /* End of function usb_hvnd_example() */
/***********************************************************************
**
 * Function Name : class_request_set_vendor
 * Description : Send Vendor Class Request (SET_VENDOR) to USB device.
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
static void class_request_set_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR; /* bRequestCode:SET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
```

```
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_set_vendor() */
/***********************************************************************
**
 * Function Name : class_request_set_vendor_no_data
 * Description : Send Vendor Class Request (SET_VENDOR_NO_DATA) to USB
device.
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
static void class_request_set_vendor_no_data (usb_instance_ctrl_t * p_ctrl,
uint8_t device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR_NO_DATA; /*
bRequestCode:SET_VENDOR_NO_DATA, bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = 0x0000;             /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_set_vendor_no_data() */
/***********************************************************************
```

```
**
 * Function Name : class_request_get_vendor
 * Description : Send Vendor Class Request (GET_VENDOR) to USB device.
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
static void class_request_get_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = 0;
    }
    setup.request_type   = GET_VENDOR; /* bRequestCode:GET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
} /* End of function class_request_get_vendor() */
/***********************************************************************
**
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
static void buffer_init (void)
```

```
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
     {
           g_buf[j][i] = (uint8_t) i;
      }
    }
} /* End of function buffer_init() */
/***********************************************************************
**
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 ***********************************************************************
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
     {
 while (1)
     {
               ;
           }
      }
    }
} /* End of function buffer_check() */
/***********************************************************************
```

```
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 ***********************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
    }
} /* End of function buffer_clear() */
/***********************************************************************
**
 * End of function usb_mcu_init
 ***********************************************************************
**/
```

### 5.2.6.41 USB PAUD (r_usb_paud)

Modules » Connectivity

This module provides a USB Peripheral Audio Device Class Driver (PAUD). It implements the USB PAUD Interface.

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

#### Detailed Description

# Overview

The r_usb_paud module combines with the r_usb_basic module to provide a USB Peripheral Audio Device Class (PAUD) driver.

### Features

The r_usb_paud module has the following key features:

- Data transfer to and from a USB host
- Response to Audio class requests
- Supports Audio 1.0 and Audio 2.0

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

## Usage Notes

### Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

- Audio 1.0

| Request | Code | Description |
|---|---|---|
| GET MIN | 0x82 | Minimum setting attribute |
| GET_MAX | 0x83 | Maximum setting attribute |
| GET_RES | 0x84 | Resolution attribute |
| SET_CUR | 0x01 | Current setting attribute |
| GET_CUR | 0x81 | Current setting attribute |

- Audio 2.0

| Request | Code | Description |
|---|---|---|
| CUR | 0x01 | Current setting attribute |
| RANGE | 0x02 | Range attribute |

Note:

For details about each device class, please refer to Audio Class specification.

### Limitations

- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB

cable.

# Examples

### USB PAUD Example

1. Example code for Audio 1.0

```
#define A10_FU_ID_0 (5)
#define A10_FU_ID_1 (8)
#define A10_EP_ADDR_OUT (0x02)
#define A10_EP_ADDR_IN (0x81)
#define A10_MUTE_0 (0)
#define A10_MUTE_1 (0x10)
#define A10_VOL_0 (0)
#define A10_VOL_MIN_0 (0)
#define A10_VOL_MAX_0 (0x80)
#define A10_VOL_RES_0 (0x40)
#define A10_VOL_1 (0)
#define A10_VOL_MIN_1 (0)
#define A10_VOL_MAX_1 (0xF0)
#define A10_VOL_RES_1 (0x80)
#define ISO_OUT_IF_NO (0x0001)
#define ISO_IN_IF_NO (0x0002)
#define ISO_IF_ALT_1 (0x0001)
#define ISO_IF_ALT_0 (0x0000)
#define ISO_OUT_MAX_PACKET_SIZE (192)
#define ISO_IN_MAX_PACKET_SIZE (200)
#define APP_BUFFER_SIZE (1024 * 200)
#define BMREQUESTTYPEDIR (0x0080U)
#define VALUE_FF00H (0xFF00)
/******************************************************************************
 * Exported global functions (to be accessed by other files)
 ******************************************************************************/
void usb_paud_example(void);
#if (BSP_CFG_RTOS == 2)
void usb_apl_callback(usb_event_info_t * p_event_info, usb_hdl_t handler, usb_onoff_t
```

```
state);
#endif
/***********************************************************************
 * Private global variables and functions
 ***********************************************************************/
static usb_paud_audio_control_group_t g_audio_control_group;
static usb_paud_audio_control_t g_audio_control[2];
static uint8_t g_paud10_snd_buf[APP_BUFFER_SIZE] = {0};
static uint8_t g_paud10_rcv_buf[ISO_OUT_MAX_PACKET_SIZE] = {0};
static uint32_t g_rcv_index = 0;
static uint32_t g_snd_index = 0;
static volatile bool g_err_flag = false;
static bool b_usb_connected = false;
static fsp_err_t process_usb_events(void);
static void process_usb_audio_class_request(void);
static void process_usb_audio_class_request_out(void);
static void process_usb_audio_class_request_in(void);
static void reset_apl_buffer_info(void);
static void handle_error(fsp_err_t err, char * err_str);
extern const usb_cfg_t g_basic0_cfg;
extern usb_instance_ctrl_t g_basic0_ctrl;
#if (BSP_CFG_RTOS == 2)
static volatile usb_event_info_t * p_usb_event = NULL;
#else
static usb_event_info_t event_info = {0};
static volatile usb_event_info_t * p_usb_event = &event_info;
#endif
#if (BSP_CFG_RTOS == 2)
extern QueueHandle_t g_queue;
#endif
/***************************************************************************************
*******************************
 * Function Name : usb_paud_example
 * Description : Peripheral Audio application main process
```

```
 * Arguments : none
 * Return value : none
 ***********************************************************************************************
*****************************/
void usb_paud_example (void)
{
 fsp_pack_version_t version = {RESET_VALUE};
 fsp_err_t          err    = FSP_SUCCESS;
#if (BSP_CFG_RTOS == 2)
    BaseType_t err_queue = pdFALSE;
#endif
 /* Version get API for FLEX pack information */
 R_FSP_VersionGet(&version);
 /* Audio Class parameter settings */
    g_audio_control[0].fu_id      = A10_FU_ID_0;
    g_audio_control[0].mute       = A10_MUTE_0;
    g_audio_control[0].volume    = A10_VOL_0;
    g_audio_control[0].volume_min = A10_VOL_MIN_0;
    g_audio_control[0].volume_max = A10_VOL_MAX_0;
    g_audio_control[0].volume_res = A10_VOL_RES_0;
    g_audio_control[0].ep_addr    = A10_EP_ADDR_OUT;
    g_audio_control[1].fu_id      = A10_FU_ID_1;
    g_audio_control[1].mute       = A10_MUTE_1;
    g_audio_control[1].volume    = A10_VOL_1;
    g_audio_control[1].volume_min = A10_VOL_MIN_1;
    g_audio_control[1].volume_max = A10_VOL_MAX_1;
    g_audio_control[1].volume_res = A10_VOL_RES_1;
    g_audio_control[1].ep_addr    = A10_EP_ADDR_IN;
    g_audio_control_group.controls_nb = 2;
    g_audio_control_group.p_controls = &g_audio_control[0];
 /* Open USB instance */
    err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
```

```
      APP_ERR_TRAP
   }
 /* USB Opened successfully. */
 while (true)
   {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
     {
           handle_error(FSP_ERR_ABORTED, "\r\nError in sending usb event through
queue\r\n");
     }
#if (BSP_CFG_RTOS == 2)
 /* Receive message from queue */
      err_queue = xQueueReceive(g_queue, &p_usb_event, (portMAX_DELAY));
 /* Handle error */
 if (pdTRUE != err_queue)
     {
           handle_error(FSP_ERR_ABORTED, "\r\nError in receiving event through
queue\r\n");
     }
#else
 /* Obtain USB related events */
      err = R_USB_EventGet(&event_info, &event_info.event);
 /* Handle error */
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
#endif
 /* process USB events */
      err = process_usb_events();
      handle_error(err, "\r\nprocess_usb_events failed.\r\n");
   }
}
```

```
/*****************************************************************************************
*****************************
 * Function Name : process_usb_events
 * Description : Function processes USB events
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation
 *****************************************************************************************
*****************************/
fsp_err_t process_usb_events (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* USB event received */
 switch (p_usb_event->event)
    {
 case USB_STATUS_CONFIGURED:
     {
           b_usb_connected = true;
 break;
     }
 case USB_STATUS_READ_COMPLETE:
     {
 if ((FSP_SUCCESS == p_usb_event->status) || (FSP_ERR_USB_SIZE_SHORT ==
p_usb_event->status))
     {
 if ((g_rcv_index + p_usb_event->data_size) < APP_BUFFER_SIZE)
     {
                 memcpy(&g_paud10_snd_buf[g_rcv_index], g_paud10_rcv_buf,
p_usb_event->data_size);
     }
              g_rcv_index += p_usb_event->data_size;
 if ((g_rcv_index + ISO_OUT_MAX_PACKET_SIZE) < APP_BUFFER_SIZE)
     {
 if (b_usb_connected)
```

```
     {
                        err = R_USB_Read(&g_basic0_ctrl, g_paud10_rcv_buf,
ISO_OUT_MAX_PACKET_SIZE, USB_CLASS_PAUD);
     }
 /* Handle error */
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
     }
 break;
     }
 case USB_STATUS_WRITE_COMPLETE:
     {
 if (FSP_SUCCESS == p_usb_event->status)
     {
               g_snd_index += ISO_IN_MAX_PACKET_SIZE;
 if (g_snd_index >= APP_BUFFER_SIZE)
     {
                  g_snd_index = 0;
     }
 else
     {
 if (b_usb_connected)
     {
                        err = R_USB_Write(&g_basic0_ctrl,
                                          &g_paud10_snd_buf[g_snd_index],
                                          ISO_IN_MAX_PACKET_SIZE,
 USB_CLASS_PAUD);
     }
 /* Handle error */
 if (FSP_SUCCESS != err)
     {
```

```
        APP_ERR_TRAP
    }
    }
    }
break;
    }
case USB_STATUS_REQUEST:       /* Receive Class Request */
    {
          process_usb_audio_class_request();
break;
    }
case USB_STATUS_REQUEST_COMPLETE:
    {
if (!(USB_HOST_TO_DEV & p_usb_event->setup.request_type) &&
              !(USB_STANDARD & p_usb_event->setup.request_type))
    {
if (USB_SET_INTERFACE == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
/* Streaming (OUT direction) IF ? */
if (ISO_OUT_IF_NO == p_usb_event->setup.request_index)
    {
if (ISO_IF_ALT_1 == p_usb_event->setup.request_value)
    {
if (b_usb_connected)
    {
                           err = R_USB_Read(&g_basic0_ctrl,
                                            g_paud10_rcv_buf,
                                            ISO_OUT_MAX_PACKET_SIZE,
USB_CLASS_PAUD);
    }
/* Handle error */
if (FSP_SUCCESS != err)
    {
        APP_ERR_TRAP
```

```
     }
     }
if (ISO_IF_ALT_0 == p_usb_event->setup.request_value)
     {
/* Do nothing */
     }
     }
/* Streaming (IN direction) IF ? */
if (ISO_IN_IF_NO == p_usb_event->setup.request_index)
     {
if (ISO_IF_ALT_1 == p_usb_event->setup.request_value)
     {
if (b_usb_connected)
     {
                        err = R_USB_Write(&g_basic0_ctrl,
                                                g_paud10_snd_buf,
                                              ISO_IN_MAX_PACKET_SIZE,
USB_CLASS_PAUD);
     }
/* Handle error */
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
if (ISO_IF_ALT_0 == p_usb_event->setup.request_value)
     {
/* Do nothing */
     }
     }
     }
     }
break;
     }
```

```
 case USB_STATUS_DETACH:
 case USB_STATUS_SUSPEND:
      {
            b_usb_connected = false;
 /*Clear the Streamg Data Buffer information */
            reset_apl_buffer_info();
 break;
      }
 case USB_STATUS_RESUME:
      {
            b_usb_connected = true;
 break;
      }
 default:
      {
 break;
      }
    }
 return err;
}
/******************************************************************************
*****************************
 * Function Name : process_usb_audio_class_request
 * Description : Function processes USB Audio Class Request events
 * Arguments : none
 * Return value : none
 ******************************************************************************
*****************************/
static void process_usb_audio_class_request (void)
{
 if (USB_HOST_TO_DEV == (BMREQUESTTYPEDIR & p_usb_event->setup.request_type))
    {
 /* Data Stage is out-direction. */
       process_usb_audio_class_request_out();
```

```
    }
 else if (USB_DEV_TO_HOST & p_usb_event->setup.request_type)
    {
 /* Data Stage is in-direction. */
       process_usb_audio_class_request_in();
    }
 else
    {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
    }
}
/******************************************************************************
*****************************
 * Function Name : process_usb_audio_class_request_out
 * Description : Function processes USB Audio Class Request events
 * Arguments : none
 * Return value : none
 ******************************************************************************
*****************************/
static void process_usb_audio_class_request_out (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint32_t i;
    uint32_t unit_id;
 if (USB_ENDPOINT & p_usb_event->setup.request_type)
    {
 if (USB_PAUD_SET_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
      {
 if (USB_PAUD_CLOCK_FREQUENCY_CONTROL == p_usb_event->setup.request_value)
      {
 /* SET_CUR (Sampling Frequency) */
 if (A10_EP_ADDR_OUT == p_usb_event->setup.request_index)
      {
```

```
                err = R_USB_PeriControlDataGet(&g_basic0_ctrl,
                                               (uint8_t *)
&g_audio_control_group.p_controls[
                                                   0].sam_freq,
                                               p_usb_event->setup.request_length)
;
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
      }
 else if (A10_EP_ADDR_IN == p_usb_event->setup.request_index)
      {
                err = R_USB_PeriControlDataGet(&g_basic0_ctrl,
                                               (uint8_t *)
&g_audio_control_group.p_controls[
                                                   1].sam_freq,
                                               p_usb_event->setup.request_length)
;
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
      }
 else
      {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
      }
 else
      {
```

```
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
      }
 else
      {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
    }
 else if (USB_PAUD_SET_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
    {
       unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
       unit_id = unit_id >> 8;
 for (i = 0; i < g_audio_control_group.controls_nb; i++)
      {
 if (unit_id == g_audio_control_group.p_controls[i].fu_id)
      {
 if (USB_PAUD_MUTE_CONTROL == p_usb_event->setup.request_value)
      {
                  err = R_USB_PeriControlDataGet(&g_basic0_ctrl,
                                                 (uint8_t *)
&g_audio_control_group.p_controls[
                                                     i].mute,
                                                 p_usb_event->setup.request_length)
;
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
 break;
      }
 if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
      {
```

```
                    err = R_USB_PeriControlDataGet(&g_basic0_ctrl,
                                                   (uint8_t *)
&g_audio_control_group.p_controls[
                                                       i].volume,
                                                   p_usb_event->setup.request_length)
;
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
 break;
      }
      }
      }
    }
 else
    {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
    }
}
/**********************************************************************************
*****************************
 * Function Name : process_usb_audio_class_request_in
 * Description : Function processes USB Audio Class Request events
 * Arguments : none
 * Return value : none
 **********************************************************************************
*****************************/
static void process_usb_audio_class_request_in (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint32_t i;
    uint32_t unit_id;
```

```
    uint32_t size;
 if (USB_ENDPOINT & p_usb_event->setup.request_type)
    {
 if (USB_PAUD_GET_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
      {
 if (USB_PAUD_CLOCK_FREQUENCY_CONTROL == p_usb_event->setup.request_value)
      {
 /* GET_CUR (Sampling Frequency) */
 if (A10_EP_ADDR_OUT == p_usb_event->setup.request_index)
      {
                 err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                (uint8_t *)
&g_audio_control_group.p_controls[
                                                    0].sam_freq,
                                                p_usb_event->setup.request_length)
;
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
      }
 else if (A10_EP_ADDR_IN == p_usb_event->setup.request_index)
      {
                 err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                (uint8_t *)
&g_audio_control_group.p_controls[
                                                    1].sam_freq,
                                                p_usb_event->setup.request_length)
;
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
```

```
     }
     }
else
     {
/* STALL */
R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
     }
     }
else
     {
/* STALL */
R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
     }
     }
else
     {
/* STALL */
R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
     }
   }
/* GET_CUR (MIN/MAX/RES)*/
else if (USB_PAUD_GET_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
   {
      unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
      unit_id = unit_id >> 8;
for (i = 0; i < g_audio_control_group.controls_nb; i++)
     {
if (unit_id == g_audio_control_group.p_controls[i].fu_id)
     {
if (USB_PAUD_MUTE_CONTROL == p_usb_event->setup.request_value)
     {
if (p_usb_event->setup.request_length <
sizeof(g_audio_control_group.p_controls[i].
                              mute))
```

```
     {
                        size = p_usb_event->setup.request_length;
     }
else
     {
                        size =
sizeof(g_audio_control_group.p_controls[i].
                                    mute);
     }
                  err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                 (uint8_t *)
&g_audio_control_group.p_controls[
                                                     i].mute,
                                                 size);
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
break;
     }
if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
if (p_usb_event->setup.request_length <
sizeof(g_audio_control_group.p_controls[i].
                                volume))
     {
                        size = p_usb_event->setup.request_length;
     }
else
     {
                        size =
sizeof(g_audio_control_group.p_controls[i].
                                    volume);
     }
```

```
                err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                               (uint8_t *)
&g_audio_control_group.p_controls[
                                                   i].volume,
                                               size);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
   }
 /* GET_MIN */
 else if (USB_PAUD_GET_MIN == (p_usb_event->setup.request_type & USB_BREQUEST))
   {
      unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
      unit_id = unit_id >> 8;
 for (i = 0; i < g_audio_control_group.controls_nb; i++)
     {
 if (unit_id == g_audio_control_group.p_controls[i].fu_id)
     {
 if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
 if (p_usb_event->setup.request_length <
 sizeof(g_audio_control_group.p_controls[i].
                            volume_min))
     {
                    size = p_usb_event->setup.request_length;
     }
 else
     {
                    size =
```

```
sizeof(g_audio_control_group.p_controls[i].
                                volume_min);
     }
                 err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                (uint8_t *)
&g_audio_control_group.p_controls[
                                                    i].volume_min,
                                                size);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
   }
 /* GET_MAX */
 else if (USB_PAUD_GET_MAX == (p_usb_event->setup.request_type & USB_BREQUEST))
   {
      unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
      unit_id = unit_id >> 8;
 for (i = 0; i < g_audio_control_group.controls_nb; i++)
     {
 if (unit_id == g_audio_control_group.p_controls[i].fu_id)
     {
 if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
 if (p_usb_event->setup.request_length <
 sizeof(g_audio_control_group.p_controls[i].
                            volume_max))
     {
                     size = p_usb_event->setup.request_length;
     }
```

```
else
     {
                         size =
sizeof(g_audio_control_group.p_controls[i].
                                    volume_max);
     }
                  err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                 (uint8_t *)
&g_audio_control_group.p_controls[
                                                     i].volume_max,
                                                 size);
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
break;
     }
     }
     }
   }
/* GET_RES */
else if (USB_PAUD_GET_RES == (p_usb_event->setup.request_type & USB_BREQUEST))
   {
      unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
      unit_id = unit_id >> 8;
for (i = 0; i < g_audio_control_group.controls_nb; i++)
     {
if (unit_id == g_audio_control_group.p_controls[i].fu_id)
     {
if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
if (p_usb_event->setup.request_length <
sizeof(g_audio_control_group.p_controls[i].
                               volume_res))
```

```
     {
                        size = p_usb_event->setup.request_length;
     }
 else
     {
                        size =
 sizeof(g_audio_control_group.p_controls[i].
                                   volume_res);
     }
                  err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                 (uint8_t *)
&g_audio_control_group.p_controls[
                                                     i].volume_res,
                                                 size);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
   }
 else
   {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
   }
}
/***********************************************************************************
*********************************
 * @brief Clear the Streamg Data Buffer information.
 * @param[in] None
 * @retval None
```

```
 ***********************************************************************************************
*********************************/
static void reset_apl_buffer_info (void)
{
    memset(g_paud10_snd_buf, 0, sizeof(g_paud10_snd_buf));
    g_rcv_index = 0;
    g_snd_index = 0;
}
#if (BSP_CFG_RTOS == 2)
/***********************************************************************************************
*****************************
 * Function Name : usb_apl_callback
 * Description : This function is callback for USB
 * Arguments : usb_event_info_t *p_event_info
 * : usb_hdl_t handler
 * : usb_onoff_t on_off
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation
 ***********************************************************************************************
*****************************/
void usb_apl_callback (usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t state)
{
 FSP_PARAMETER_NOT_USED(handler);
 FSP_PARAMETER_NOT_USED(state);
 /* Send event received to queue */
 if (pdTRUE != (xQueueSend(g_queue, (const void *) &p_event_info, (TickType_t)
(RESET_VALUE))))
    {
       g_err_flag = true;
    }
}
#endif
/***********************************************************************************************
```

```
*******************************
 * Function Name : handle_error
 * Description : Print and traps error
 * Arguments : fsp_err_t err
 * : char * err_str
 * Return value : none
 ***********************************************************************************
*******************************/
static void handle_error (fsp_err_t err, char * err_str)
{
 FSP_PARAMETER_NOT_USED(err_str);
 if (FSP_SUCCESS != err)
    {
 if (FSP_SUCCESS != R_USB_Close(&g_basic0_ctrl))
      {
 /* R_USB_Close API Failed */
         APP_ERR_TRAP
      }
       APP_ERR_TRAP
    }
}
```

1. Example code for Audio 2.0

```
#define USB_PAUD_CLOCK_SOURCE_UNIT (0x1000U) /* Clock Source Unit ID */
#define A20_CS_ID (0x10)
#define A20_SAMPLING_FREQUENCY (48000)
#define A20_FU_ID_0 (5)
#define A20_FU_ID_1 (8)
#define A20_MUTE (0)
#define A20_VOL_MIN_0 (0)
#define A20_VOL_MAX_0 (100)
#define A20_SET_VOL_0 (50)
#define A20_VOL_MIN_1 (0)
```

```
#define A20_VOL_MAX_1 (200)
#define A20_SET_VOL_1 (100)
#define ISO_OUT_IF_NO (0x0001)
#define ISO_IN_IF_NO (0x0002)
#define ISO_IF_ALT_1 (0x0001)
#define ISO_IF_ALT_0 (0x0000)
#define ISO_OUT_MAX_PACKET_SIZE (192)
#define ISO_IN_MAX_PACKET_SIZE (200)
#define APP_BUFFER_SIZE (1024 * 200)
#define RANGE_DATA_TOP2BYTES (0x0001)
#define BMREQUESTTYPEDIR (0x0080U)
#define VAULE_FFH (0xFF)
#define VALUE_FF00H (0xFF00)
/***********************************************************************
 * Exported global functions (to be accessed by other files)
 ***********************************************************************/
void usb_paud_example(void);
#if (BSP_CFG_RTOS == 2)
void usb_apl_callback(usb_event_info_t * p_event_info, usb_hdl_t handler, usb_onoff_t
state);
#endif
/***********************************************************************
 * Private global variables and functions
 ***********************************************************************/
static usb_paud_audio_control_group_t g_audio20_control_group;
static usb_paud_audio_control_t g_audio20_control[2];
static uint8_t g_paud20_snd_buf[APP_BUFFER_SIZE] = {0};
static uint8_t g_paud20_rcv_buf[ISO_OUT_MAX_PACKET_SIZE] = {0};
static uint32_t g_rcv_index = 0;
static uint32_t g_snd_index = 0;
static volatile bool g_err_flag = false;
static bool b_usb_connected = false;
static fsp_err_t process_usb_events(void);
static void process_usb_audio_class_request(void);
```

```
static void reset_apl_buffer_info(void);
static void handle_error(fsp_err_t err, char * err_str);
extern const usb_cfg_t g_basic0_cfg;
extern usb_instance_ctrl_t g_basic0_ctrl;
#if (BSP_CFG_RTOS == 2)
static volatile usb_event_info_t * p_usb_event = NULL;
#else
static usb_event_info_t event_info = {0};
static volatile usb_event_info_t * p_usb_event = &event_info;
#endif
#if (BSP_CFG_RTOS == 2)
extern QueueHandle_t g_queue;
#endif
/***********************************************************************************************
*****************************
 * Function Name : usb_paud_example
 * Description : Peripheral Audio application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************************************
*****************************/
void usb_paud_example (void)
{
 fsp_pack_version_t version = {RESET_VALUE};
 fsp_err_t          err    = FSP_SUCCESS;
#if (BSP_CFG_RTOS == 2)
    BaseType_t err_queue = pdFALSE;
#endif
 /* Version get API for FLEX pack information */
 R_FSP_VersionGet(&version);
 /* Audio Class parameter settings */
    g_audio20_control[0].cs_id      = A20_CS_ID;
    g_audio20_control[0].sam_freq   = A20_SAMPLING_FREQUENCY;
    g_audio20_control[0].fu_id      = A20_FU_ID_0;
```

```
    g_audio20_control[0].mute       = A20_MUTE;
    g_audio20_control[0].volume_min = A20_VOL_MIN_0;
    g_audio20_control[0].volume_max = A20_VOL_MAX_0;
    g_audio20_control[0].volume    = A20_SET_VOL_0;
    g_audio20_control[1].cs_id      = A20_CS_ID;
    g_audio20_control[1].sam_freq   = A20_SAMPLING_FREQUENCY;
    g_audio20_control[1].fu_id      = A20_FU_ID_1;
    g_audio20_control[1].mute       = A20_MUTE;
    g_audio20_control[1].volume_min = A20_VOL_MIN_1;
    g_audio20_control[1].volume_max = A20_VOL_MAX_1;
    g_audio20_control[1].volume    = A20_SET_VOL_1;
    g_audio20_control_group.controls_nb = 2;
    g_audio20_control_group.p_controls = &g_audio20_control[0];
 /* Open USB instance */
    err = R_USB_Open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
       APP_ERR_TRAP
    }
 /* USB Opened successfully. */
 while (true)
    {
 /* Handle error if queue send fails*/
 if (true == g_err_flag)
     {
           handle_error(FSP_ERR_ABORTED, "\r\nError in sending usb event through
queue\r\n");
     }
#if (BSP_CFG_RTOS == 2)
 /* Receive message from queue */
       err_queue = xQueueReceive(g_queue, &p_usb_event, (portMAX_DELAY));
 /* Handle error */
 if (pdTRUE != err_queue)
     {
```

```
            handle_error(FSP_ERR_ABORTED, "\r\nError in receiving event through
queue\r\n");
      }
#else
 /* Obtain USB related events */
       err = R_USB_EventGet(&event_info, &event_info.event);
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
#endif
 /* process USB events */
       err = process_usb_events();
       handle_error(err, "\r\nprocess_usb_events failed.\r\n");
    }
}
/******************************************************************************
*****************************
 * Function Name : process_usb_events
 * Description : Function processes USB events
 * Arguments : none
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation
 ******************************************************************************
*****************************/
fsp_err_t process_usb_events (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* USB event received */
 switch (p_usb_event->event)
    {
 case USB_STATUS_CONFIGURED:
      {
```

```
            b_usb_connected = true;
 break;
      }
 case USB_STATUS_READ_COMPLETE:
      {
 if ((FSP_SUCCESS == p_usb_event->status) || (FSP_ERR_USB_SIZE_SHORT ==
p_usb_event->status))
      {
 if ((g_rcv_index + p_usb_event->data_size) < APP_BUFFER_SIZE)
      {
                  memcpy(&g_paud20_snd_buf[g_rcv_index], g_paud20_rcv_buf,
p_usb_event->data_size);
      }
               g_rcv_index += p_usb_event->data_size;
 if ((g_rcv_index + ISO_OUT_MAX_PACKET_SIZE) < APP_BUFFER_SIZE)
      {
 if (b_usb_connected)
      {
                      err = R_USB_Read(&g_basic0_ctrl, g_paud20_rcv_buf,
ISO_OUT_MAX_PACKET_SIZE, USB_CLASS_PAUD);
      }
 /* Handle error */
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
      }
      }
 break;
      }
 case USB_STATUS_WRITE_COMPLETE:
      {
 if (FSP_SUCCESS == p_usb_event->status)
      {
```

```
                g_snd_index += ISO_IN_MAX_PACKET_SIZE;
if (g_snd_index >= APP_BUFFER_SIZE)
     {
                   g_snd_index = 0;
     }
else
     {
if (b_usb_connected)
     {
                        err = R_USB_Write(&g_basic0_ctrl,
                                          &g_paud20_snd_buf[g_snd_index],
                                          ISO_IN_MAX_PACKET_SIZE,
USB_CLASS_PAUD);
     }
/* Handle error */
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
     }
break;
     }
case USB_STATUS_REQUEST:       /* Receive Class Request */
     {
           process_usb_audio_class_request();
break;
     }
case USB_STATUS_REQUEST_COMPLETE:
     {
if (!(USB_HOST_TO_DEV & p_usb_event->setup.request_type) &&
                !(USB_STANDARD & p_usb_event->setup.request_type))
     {
if (USB_SET_INTERFACE == (p_usb_event->setup.request_type & USB_BREQUEST))
```

```
     {
/* Streaming (OUT direction) IF ? */
if (ISO_OUT_IF_NO == p_usb_event->setup.request_index)
     {
if (ISO_IF_ALT_1 == p_usb_event->setup.request_value)
     {
if (b_usb_connected)
     {
                              err = R_USB_Read(&g_basic0_ctrl,
                                               g_paud20_rcv_buf,
                                               ISO_OUT_MAX_PACKET_SIZE,
USB_CLASS_PAUD);
     }
/* Handle error */
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
if (ISO_IF_ALT_0 == p_usb_event->setup.request_value)
     {
/* Do nothing */
     }
     }
/* Streaming (IN direction) IF ? */
if (ISO_IN_IF_NO == p_usb_event->setup.request_index)
     {
if (ISO_IF_ALT_1 == p_usb_event->setup.request_value)
     {
if (b_usb_connected)
     {
                         err = R_USB_Write(&g_basic0_ctrl,
                                               g_paud20_snd_buf,
                                           ISO_IN_MAX_PACKET_SIZE,
```

```
USB_CLASS_PAUD);
     }
/* Handle error */
if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
     }
if (ISO_IF_ALT_0 == p_usb_event->setup.request_value)
     {
/* Do nothing */
     }
     }
     }
     }
break;
     }
case USB_STATUS_DETACH:
case USB_STATUS_SUSPEND:
     {
           b_usb_connected = false;
/*Clear the Streamg Data Buffer information */
           reset_apl_buffer_info();
break;
     }
case USB_STATUS_RESUME:
     {
           b_usb_connected = true;
break;
     }
default:
     {
break;
     }
```

```
    }
 return err;
}
/**********************************************************************************************
*****************************
 * Function Name : process_usb_audio_class_request
 * Description : Function processes USB Audio Class Request events
 * Arguments : none
 * Return value : none
 **********************************************************************************************
*****************************/
static void process_usb_audio_class_request (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint32_t i;
    uint32_t unit_id;
    uint32_t freq;
    uint8_t * p_range;
    uint8_t   volume_range[8];
    uint8_t   smpl_freq_range[14];
 if (USB_HOST_TO_DEV == (BMREQUESTTYPEDIR & p_usb_event->setup.request_type))
    {
 if (USB_PAUD_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
      {
 if ((USB_PAUD_CLOCK_FREQUENCY_CONTROL == p_usb_event->setup.request_value) &&
                (USB_PAUD_CLOCK_SOURCE_UNIT == p_usb_event->setup.request_index))
      {
         unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
         unit_id = unit_id >> 8;
                err = R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &freq,
p_usb_event->setup.request_length);
 if (FSP_SUCCESS == err)
      {
 for (i = 0; i < g_audio20_control_group.controls_nb; i++)
```

```
     {
if (unit_id ==
                          g_audio20_control_group.p_controls[i].cs_id)
     {
if (0 ==
                             g_audio20_control_group.p_controls[i].
                             sam_freq)
     {
                             g_audio20_control_group.p_controls[i].
                             sam_freq = freq;
     }
     }
     }
     }
/* Handle error */
else
     {
        APP_ERR_TRAP
     }
     }
else if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
        unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
        unit_id = unit_id >> 8;
for (i = 0; i < g_audio20_control_group.controls_nb; i++)
     {
if (unit_id ==
                       g_audio20_control_group.p_controls[i].fu_id)
     {
                    err = R_USB_PeriControlDataGet(&g_basic0_ctrl,
                                                   (uint8_t *)
&g_audio20_control_group.p_controls[
                                                     i].volume,
```

```
p_usb_event->setup.request_length);
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
 break;
      }
      }
      }
 else
      {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
      }
 else
      {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
    }
 else if (USB_DEV_TO_HOST & p_usb_event->setup.request_type)
    {
 if (USB_PAUD_CUR == (p_usb_event->setup.request_type & USB_BREQUEST))
      {
 if ((USB_PAUD_CLOCK_FREQUENCY_CONTROL == p_usb_event->setup.request_value) &&
                (USB_PAUD_CLOCK_SOURCE_UNIT == p_usb_event->setup.request_index))
      {
         unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
         unit_id = unit_id >> 8;
 for (i = 0; i < g_audio20_control_group.controls_nb; i++)
      {
 if (unit_id ==
                       g_audio20_control_group.p_controls[i].cs_id)
```

```
     {
                    err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                   (uint8_t *)
&g_audio20_control_group.p_controls[
                                                      i].sam_freq,

p_usb_event->setup.request_length);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
 else if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
        unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
        unit_id = unit_id >> 8;
 for (i = 0; i < g_audio20_control_group.controls_nb; i++)
     {
 if (unit_id ==
                       g_audio20_control_group.p_controls[i].fu_id)
     {
                    err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                   (uint8_t *)
&g_audio20_control_group.p_controls[
                                                     i].volume,

p_usb_event->setup.request_length);
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
```

```
 break;
      }
      }
      }
 else if (USB_PAUD_MUTE_CONTROL == p_usb_event->setup.request_value)
      {
         unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
         unit_id = unit_id >> 8;
 for (i = 0; i < g_audio20_control_group.controls_nb; i++)
      {
 if (unit_id ==
                       g_audio20_control_group.p_controls[i].fu_id)
      {
                   err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                  (uint8_t *)
&g_audio20_control_group.p_controls[
                                                    i].mute,

p_usb_event->setup.request_length);
 if (FSP_SUCCESS != err)
      {
         APP_ERR_TRAP
      }
 break;
      }
      }
      }
 else
      {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
      }
      }
 else if (USB_PAUD_RANGE == (p_usb_event->setup.request_type & USB_BREQUEST))
```

```
     {
 if ((USB_PAUD_CLOCK_FREQUENCY_CONTROL == p_usb_event->setup.request_value) &&
                (USB_PAUD_CLOCK_SOURCE_UNIT == p_usb_event->setup.request_index))
     {
        unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
        unit_id = unit_id >> 8;
 for (i = 0; i < g_audio20_control_group.controls_nb; i++)
     {
 if (unit_id ==
                       g_audio20_control_group.p_controls[i].cs_id)
     {
                       p_range = smpl_freq_range;
                       *p_range++ = (uint8_t) (RANGE_DATA_TOP2BYTES & VAULE_FFH);
                       *p_range++ = (uint8_t) ((RANGE_DATA_TOP2BYTES >> 8) &
VAULE_FFH);
                       *p_range++ =
                           (uint8_t) ((g_audio20_control_group.p_controls[i].
                                sam_freq) & VAULE_FFH);
                       *p_range++ =
                           (uint8_t) ((g_audio20_control_group.p_controls[i].
                                       sam_freq >> 8) & VAULE_FFH);
                       *p_range++ =
                           (uint8_t) ((g_audio20_control_group.p_controls[i].
                                       sam_freq >> 16) & VAULE_FFH);
                       *p_range++ =
                           (uint8_t) ((g_audio20_control_group.p_controls[i].
                                       sam_freq >> 24) & VAULE_FFH);
                       *p_range++ =
                           (uint8_t) (g_audio20_control_group.p_controls[i].
                                      sam_freq & VAULE_FFH);
                       *p_range++ =
                           (uint8_t) ((g_audio20_control_group.p_controls[i].
                                       sam_freq >>
                                       8) & VAULE_FFH);
```

```
                        *p_range++ =
                            (uint8_t) ((g_audio20_control_group.p_controls[i].
                                        sam_freq >> 16) & VAULE_FFH);
                        *p_range++ =
                            (uint8_t) ((g_audio20_control_group.p_controls[i].
                                        sam_freq >> 24) & VAULE_FFH);
                        *p_range++ = (uint8_t) 0x00;
                        *p_range++ = (uint8_t) 0x00;
                        *p_range++ = (uint8_t) 0x00;
                        *p_range++ = (uint8_t) 0x00;
 if (sizeof(smpl_freq_range) < p_usb_event->setup.request_length)
     {
                            err =
 R_USB_PeriControlDataSet(&g_basic0_ctrl, smpl_freq_range, sizeof(smpl_freq_range));
     }
 else
     {
                    err = R_USB_PeriControlDataSet(&g_basic0_ctrl,
                                                   smpl_freq_range,

p_usb_event->setup.request_length);
     }
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
 else if (USB_PAUD_VOLUME_CONTROL == p_usb_event->setup.request_value)
     {
        unit_id = (uint32_t) (p_usb_event->setup.request_index & VALUE_FF00H);
        unit_id = unit_id >> 8;
```

```
for (i = 0; i < g_audio20_control_group.controls_nb; i++)
     {
if (unit_id ==
                      g_audio20_control_group.p_controls[i].fu_id)
     {
                      p_range = volume_range;
                      *p_range++ = (uint8_t) (RANGE_DATA_TOP2BYTES & VAULE_FFH);
                      *p_range++ = (uint8_t) ((RANGE_DATA_TOP2BYTES >> 8) &
VAULE_FFH);
                      *p_range++ =
                          (uint8_t) (g_audio20_control_group.p_controls[i].
                                     volume_min &
                                     VAULE_FFH);
                      *p_range++ =
                          (uint8_t) ((g_audio20_control_group.p_controls[i].
                                      volume_min >>
                                      8) & VAULE_FFH);
                      *p_range++ =
                          (uint8_t) (g_audio20_control_group.p_controls[i].
                                     volume_max &
                                     VAULE_FFH);
                      *p_range++ =
                          (uint8_t) ((g_audio20_control_group.p_controls[i].
                                      volume_max >>
                                      8) & VAULE_FFH);
if (1 <
                          g_audio20_control_group.p_controls[i].
                          volume_res)
     {
                        *p_range++ =
                            (uint8_t) (g_audio20_control_group.p_controls[i].
                                       volume_res &
                                      VAULE_FFH);
                          *p_range =
```

```
                                (uint8_t) ((g_audio20_control_group.p_controls[i].
                                                volume_res >> 8) & VAULE_FFH);
     }
 else
     {
                          *p_range++ = (uint8_t) (RANGE_DATA_TOP2BYTES & VAULE_FFH);
                            *p_range   = (uint8_t) ((RANGE_DATA_TOP2BYTES >> 8) &
VAULE_FFH);
     }
                        err =
 R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) volume_range, sizeof
(volume_range));
 if (FSP_SUCCESS != err)
     {
        APP_ERR_TRAP
     }
 break;
     }
     }
     }
 else
     {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
     }
     }
 else
     {
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
     }
   }
 else
   {
```

```
 /* STALL */
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_STALL);
    }
}
/***********************************************************************************************
*********************************
 * @brief Clear the Streamg Data Buffer information.
 * @param[in] None
 * @retval None
 ***********************************************************************************************
*********************************/
static void reset_apl_buffer_info (void)
{
    memset(g_paud20_snd_buf, 0, sizeof(g_paud20_snd_buf));
    g_rcv_index = 0;
    g_snd_index = 0;
}
#if (BSP_CFG_RTOS == 2)
/***********************************************************************************************
*****************************
 * Function Name : usb_apl_callback
 * Description : This function is callback for USB
 * Arguments : usb_event_info_t *p_event_info
 * : usb_hdl_t handler
 * : usb_onoff_t on_off
 * Return value : Any Other Error code apart from FSP_SUCCESS on Unsuccessful
operation
 ***********************************************************************************************
*****************************/
void usb_apl_callback (usb_event_info_t * p_event_info, usb_hdl_t handler,
usb_onoff_t state)
{
 FSP_PARAMETER_NOT_USED(handler);
 FSP_PARAMETER_NOT_USED(state);
```

```
 /* Send event received to queue */
 if (pdTRUE != (xQueueSend(g_queue, (const void *) &p_event_info, (TickType_t)
(RESET_VALUE))))
    {
       g_err_flag = true;
    }
}
#endif
/**********************************************************************************
*****************************
 * Function Name : handle_error
 * Description : Print and traps error
 * Arguments : fsp_err_t err
 * : char * err_str
 * Return value : none
 **********************************************************************************
*****************************/
static void handle_error (fsp_err_t err, char * err_str)
{
 FSP_PARAMETER_NOT_USED(err_str);
 if (FSP_SUCCESS != err)
    {
 if (FSP_SUCCESS != R_USB_Close(&g_basic0_ctrl))
      {
 /* R_USB_Close API Failed */
         APP_ERR_TRAP
      }
       APP_ERR_TRAP
    }
}
```

# Descriptor

A template for PAUD descriptors can be found in

ra/fsp/src/r_usb_paud/r_usb_paud_descriptor.c.template. Also, please be sure to use your vendor ID.

#### 5.2.6.42 USB PCDC (r_usb_pcdc)

Modules » Connectivity

This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface.

**Functions**

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

**Detailed Description**

# Overview

The r_usb_pcdc module combines with the r_usb_basic module to provide a USB Peripheral Communications Device Class (PCDC) driver. The PCDC driver conforms to Abstract Control Model of the USB Communications Device Class (CDC) specification and enables communication with a CDC host device.

**Features**

The r_usb_pcdc module has the following key features:

- Data transfer to and from a USB host
- Response to CDC class requests
- Supports CDC notifications

# Configuration

**Build Time Configurations for r_usb_pcdc**

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Connectivity > USB PCDC (r_usb_pcdc)**

This module can be added to the Stacks tab via New Stack > Connectivity > USB PCDC (r_usb_pcdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_pcdc0 | Module name. |

*Note*

*Refer to the USB (r_usb_basic) module for hardware configuration options.*

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems.

### Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

| Request | Code | Description |
|---|---|---|
| SetLineCoding | 0x20 | Sets communication line settings (bitrate, data length, parity, and stop bit length) |
| GetLineCoding | 0x21 | Acquires the communication line setting state |
| SetControlLineState | 0x22 | Set communication line control signals (RTS, DTR) |

*Note*

*For details concerning the Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.*

### Data Format of Class Requests

The data format of supported class requests is described below:

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_LINE_CODING (0x20) | 0x0000 | 0x0000 | 0x0007 | usb_pcdc_linecoding_t |
| 0xA1 | GET_LINE_CODING (0x21) | 0x0000 | 0x0000 | 0x0007 | usb_pcdc_linecoding_t |
| 0x21 | SET_CONTROL_LINE_STATE (0x22) | usb_pcdc_ctrlinestate_t | 0x0000 | 0x0000 | None |

### Class Notifications (Peripheral to Host)

The following class notifications are supported:

| Notification | Code | Description |
|---|---|---|
| SERIAL_STATE | 0x20 | Notification of serial line state |

The data types returned are as follows:

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | SERIAL_STATE (0x20) | 0x0000 | 0x0000 | 0x0002 | usb_serial_state_bitmap_t |

*Note*

*The host is notified with SERIAL_STATE whenever a change in the UART port state is detected. This driver will automatically detect overrun, parity and framing errors. A state notification is performed when a transition from normal to error state is detected.*

### Virtual COM-port Usage

When connected to a PC the CDC device can be used as a virtual COM port. After enumeration, the CDC class requests GetLineCoding and SetControlLineState are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as PuTTY. When changing settings of the serial port in the terminal application, the UART setting is propagated to the firmware via the class request SetLineCoding.

Data input (or file transmission) from the terminal app window is transmitted to the board using endpoint 2 (EP2); data from the board side is transmitted to the PC using EP1.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

### Multi Port

This driver supports simultaneous operation with Host Mass Storage Class(HMSC). If the user are using MCU that supports 2 USB modules, such as RA6M3, the user can run PCDC on one USB module and HMSC on the other. This driver does not support simultaneous operation using device classes other than HMSC.

### Limitations

- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- This driver does not support Low-speed.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

# Examples

## USB PCDC Loopback Example

The main functions of the PCDC loopback example are as follows:

1. Receives virtual UART configuration data from the host terminal
2. Loops all other received data back to the host terminal

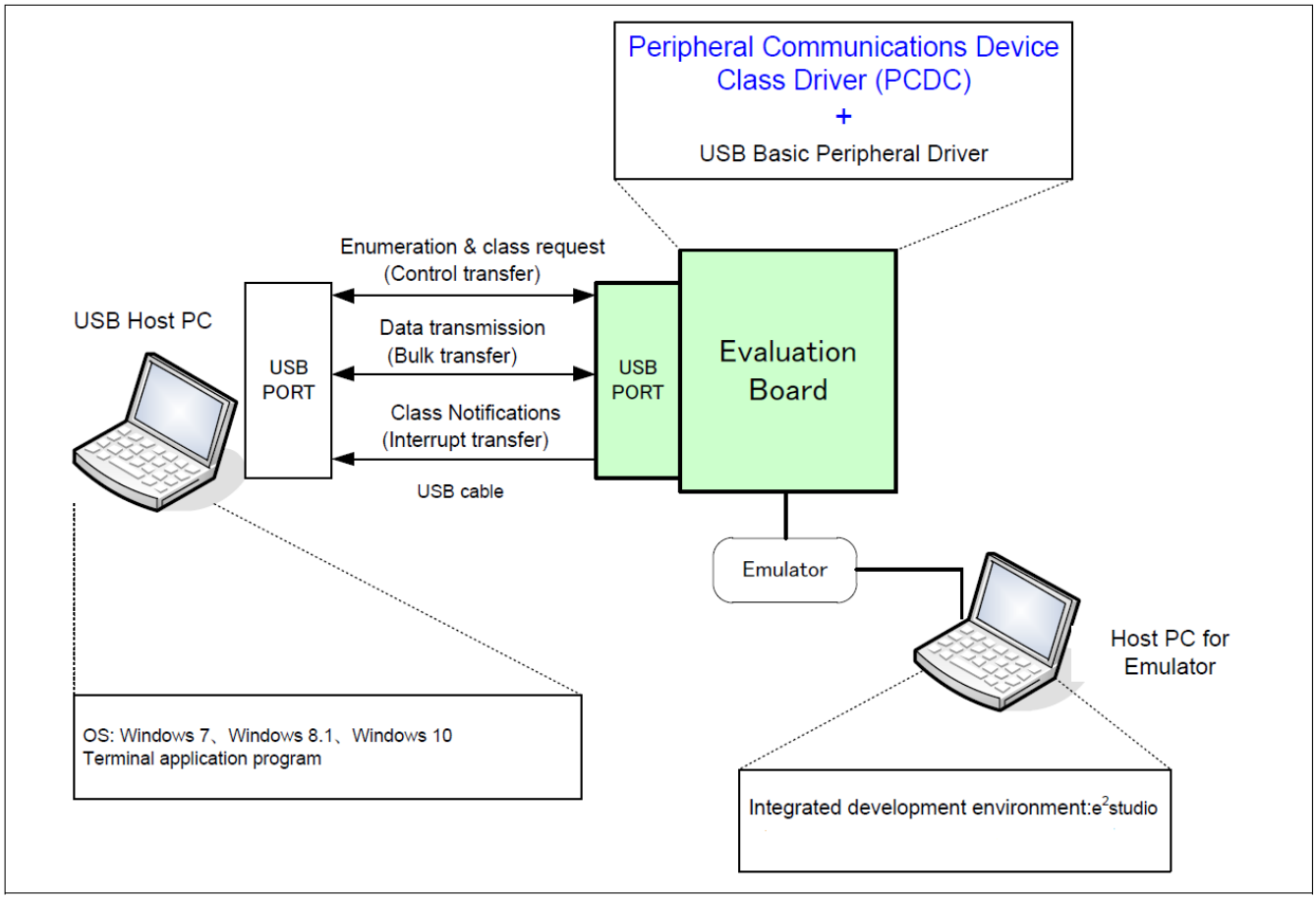



Figure 245: Example Operating Environment

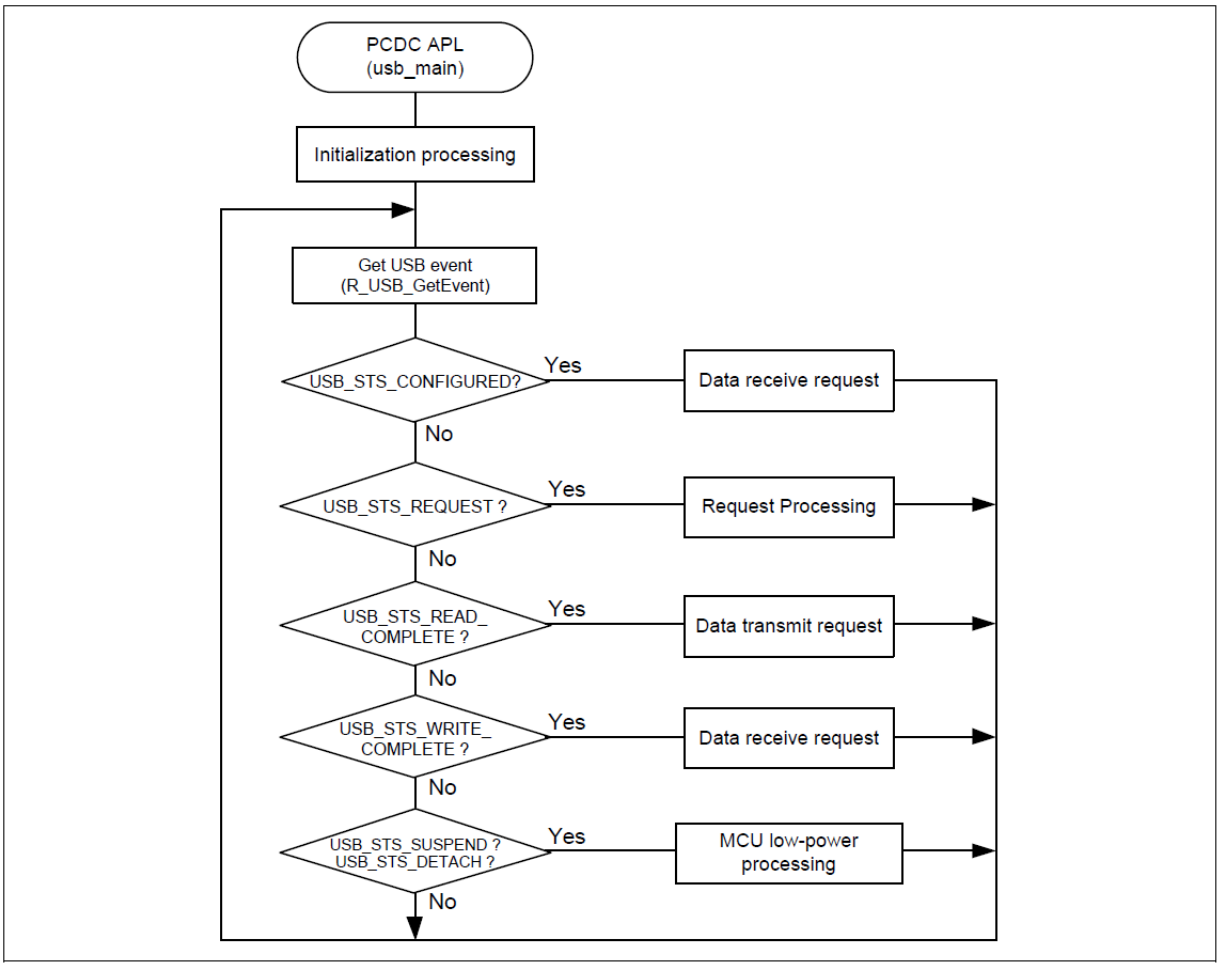



Figure 246: Main Loop processing (Echo mode)

```
#define USB_APL_YES (1U)
#define USB_APL_NO (0U)
#define APL_NUM_USB_EVENT (10U)
// #define APL_USE_BAREMETAL_CALLBACK USB_APL_NO
#define APL_USE_BAREMETAL_CALLBACK USB_APL_YES
/***********************************************************************************
 * Private global variables and functions
 ***********************************************************************************/
extern const usb_cfg_t g_basic0_cfg;
static uint8_t g_buf[DATA_LEN];
static usb_pcdc_linecoding_t g_line_coding;
extern uint8_t g_apl_device[];
extern uint8_t g_apl_configuration[];
```

```
extern uint8_t g_apl_hs_configuration[];
extern uint8_t g_apl_qualifier_descriptor[];
extern uint8_t * g_apl_string_table[];
usb_instance_ctrl_t g_basic0_ctrl;
#if (BSP_CFG_RTOS == 2)
QueueHandle_t g_apl_mbx_hdl;
#endif  /* (BSP_CFG_RTOS == 2) */
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
usb_callback_args_t g_apl_usb_event;
usb_callback_args_t g_apl_usb_event_buf[APL_NUM_USB_EVENT];
uint8_t             g_apl_usb_event_wp = 0;
uint8_t             g_apl_usb_event_rp = 0;
#endif  /* (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
/******************************************************************************
 * Exported global functions (to be accessed by other files)
 ******************************************************************************/
void usb_pcdc_example(void);
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
void usb_apl_callback(usb_callback_args_t * p_event);
#endif
#if (BSP_CFG_RTOS == 2)
void usb_apl_callback(usb_event_info_t * p_api_event, usb_hdl_t cur_task, usb_onoff_t
usb_state);
#endif
/******************************************************************************
 * Renesas Peripheral Communications Devices Class Sample Code functions
 ******************************************************************************/
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
```

```
 * Return value : none
 ***********************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
#endif /* (BSP_CFG_RTOS == 2) */
/***********************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_callback_args_t * p_event : Pointer to usb_callback_args_t
structure
 * Return value : none
 ***********************************************************************/
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
void usb_apl_callback (usb_callback_args_t * p_event)
{
    g_apl_usb_event_buf[g_apl_usb_event_wp] = *p_event;
    g_apl_usb_event_wp++;
    g_apl_usb_event_wp %= APL_NUM_USB_EVENT;
}
#endif  /* (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
/***********************************************************************
 * Function Name : usb_pcdc_example
 * Description : Peripheral CDC application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************/
void usb_pcdc_example (void)
{
    usb_event_info_t event_info;
```

```
 usb_status_t event = USB_STATUS_POWERED;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
#if (BSP_CFG_RTOS == 0) && (APL_USE_BAREMETAL_CALLBACK == USB_YES)
    g_usb_on_usb.callbackMemorySet(&g_basic0_ctrl, &g_apl_usb_event);
#endif  /* (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
    memset(&event_info, 0, sizeof(usb_event_info_t));
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 #if (APL_USE_BAREMETAL_CALLBACK == USB_YES)
       g_usb_on_usb.driverActivate(&g_basic0_ctrl);
 if (g_apl_usb_event_wp != g_apl_usb_event_rp)
     {
           event_info = g_apl_usb_event_buf[g_apl_usb_event_rp];
           g_apl_usb_event_rp++;
           g_apl_usb_event_rp %= APL_NUM_USB_EVENT;
 event = event_info.event;
      }
 #else  /* (APL_USE_BAREMETAL_CALLBACK == USB_YES) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
 #endif
#endif  /* (BSP_CFG_RTOS == 2) */
 /* Handle the received event (if any) */
 switch (event)
     {
 case USB_STATUS_CONFIGURED:
```

```
 /* Initialization complete; get data from host */
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (FSP_SUCCESS == event_info.status)
      {
 /* Initialization complete; get data from host */
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
         }
 break;
 case USB_STATUS_READ_COMPLETE:
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
      {
 /* Loop back received data to host */
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
         }
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Configure virtual UART settings */
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
         }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Send virtual UART settings back to host */
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
         }
 else
      {
```

```
 /* ACK all other status requests */
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
         }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
       }
    }
} /* End of function usb_pcdc_example() */
```

# Descriptor

A template for PCDC descriptors can be found in ra/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

### 5.2.6.43 USB PHID (r_usb_phid)

Modules » Connectivity

This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

#### Detailed Description

# Overview

The r_usb_phid module combines with the r_usb_basic module to provide a USB Peripheral Human Interface Device Class (PHID) driver. The PHID driver conforms to the USB Human Interface Device class specifications and implements communication with a HID host.

#### Features

The r_usb_phid module has the following functions:

- Data transfer to and from a USB host
- Response to HID class requests
- Response to function references from the HID host

*Note*

*This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB hosts.*

# Configuration

### Build Time Configurations for r_usb_phid

The following build time configurations are defined in fsp_cfg/r_usb_phid_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Connectivity > USB PHID (r_usb_phid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PHID (r_usb_phid).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_phid0 | Module name. |

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

| Request | Code | Description |
|---|---|---|
| Get_Report | 0x01 | Receives a report from the HID host |
| Set_Report | 0x09 | Sends a report to the HID host |
| Get_Idle | 0x02 | Receives a duration (time) from the HID host |
| Set_Idle | 0x0A | Sends a duration (time) to the HID host |
| Get_Protocol | 0x03 | Reads a protocol from the HID |

| | | |
|---|---|---|
| | | host |
| Set_Protocol | 0x0B | Sends a protocol to the HID host |
| Get_Descriptor | 0x06 | Transmits a report or HID descriptor |

The data format of supported class requests is described below:

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_REPORT (0x01) | ReportType & ReportID | Interface | ReportLength | Report |
| 0x21 | SET_REPORT (0x09) | ReportType & ReportID | Interface | ReportLength | Report |
| 0xA1 | GET_IDLE (0x02) | 0 & ReportID | Interface | 1 | Idle rate |
| 0x21 | SET_IDLE (0x0A) | Duration & ReportID | Interface | 0 | Idle rate |
| 0xA1 | GET_PROTOCOL (0x03) | 0 | Interface | 0 | 0 (Boot) or 1 (Report) |
| 0x21 | SET_PROTOCOL (0x0B) | 0 (Boot) or 1 (Report) | Interface | 0 | Not applicable |

**Limitations**

- This driver does not support USB Hi-speed mode.
- This driver does not support USB Low-speed mode.
- This driver does not support DMA transfers.
- This driver does not support simultaneous operation with USB Host device class.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

# Examples

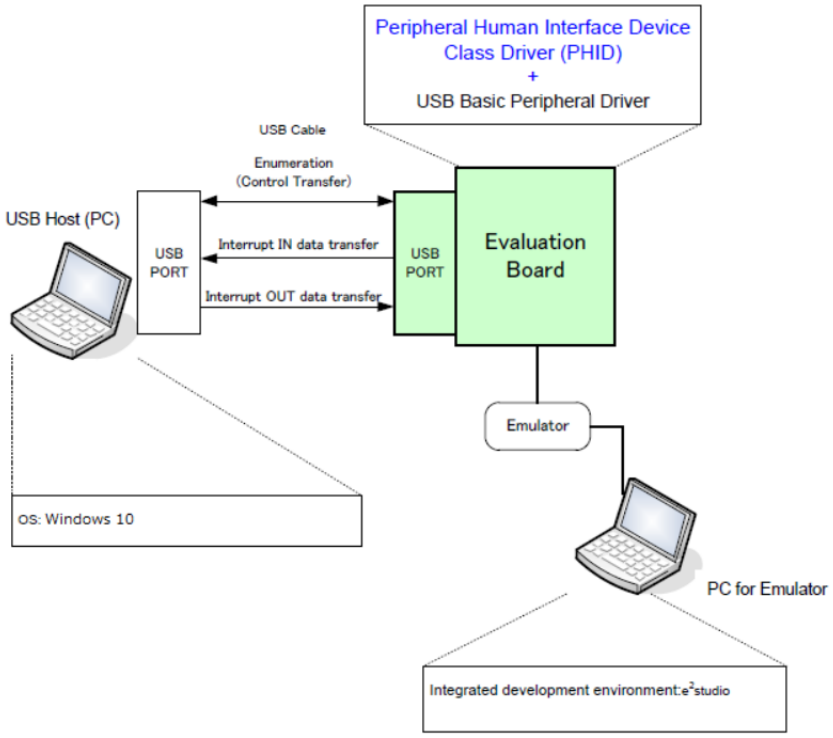



Figure 247: Example Operating Environment

### USB PHID Example

This is a minimal example for implementing PHID in a non-RTOS application.

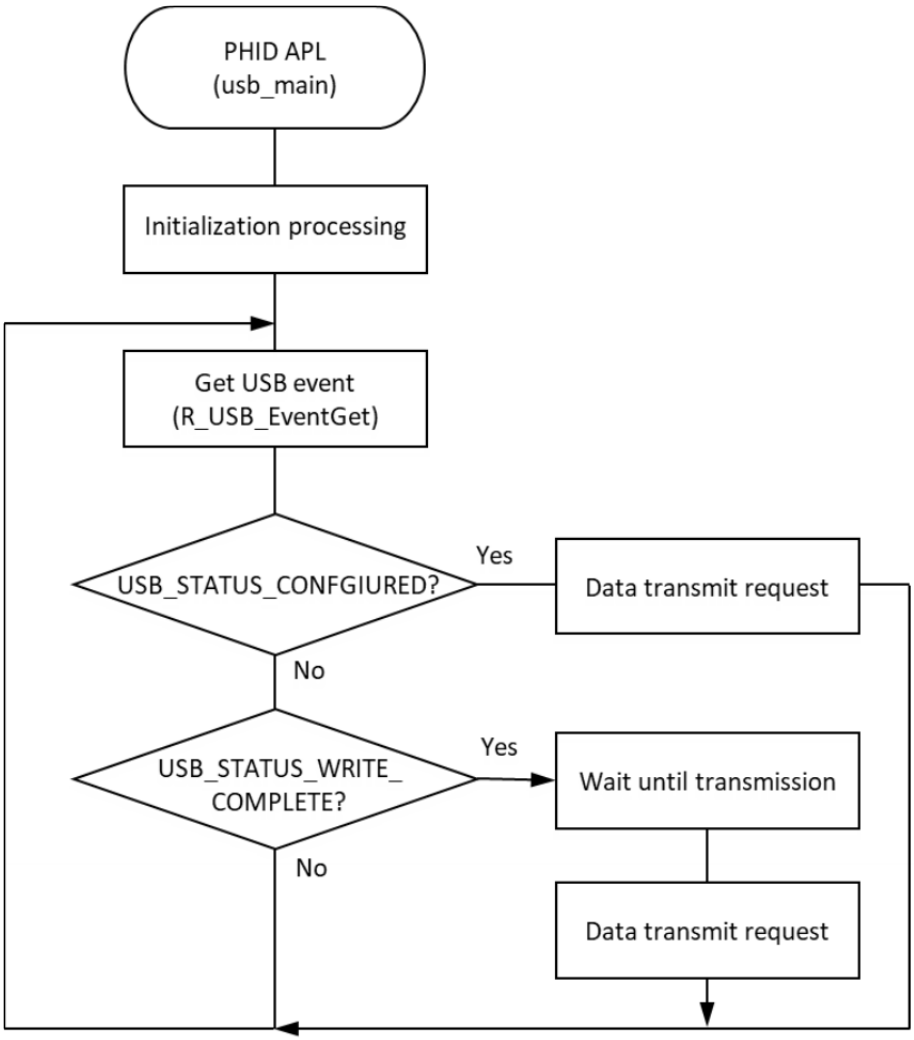



Figure 248: Main Loop processing for non-RTOS example

This is a minimal example for implementing PHID in an RTOS application.

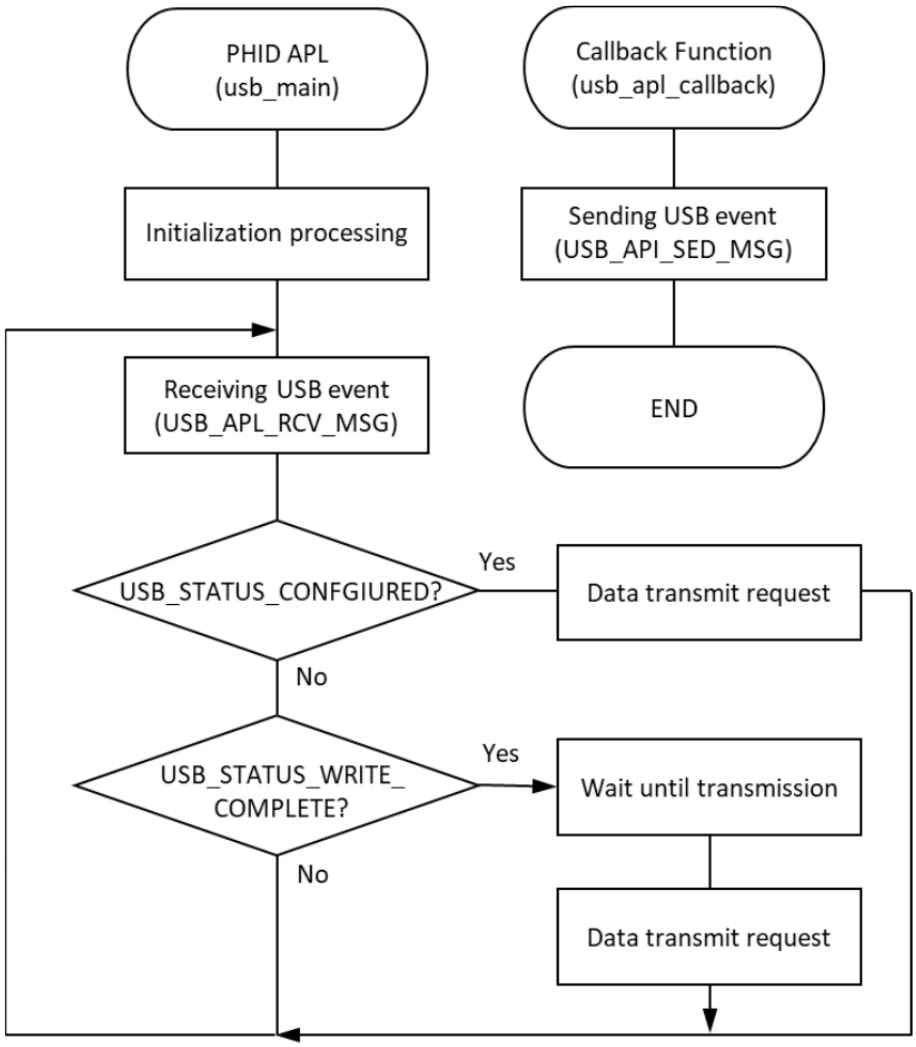



Figure 249: Main Loop processing for RTOS example

```
#define USB_RECEIVE_REPORT_DESCRIPTOR (76)
#define USB_RECEIVE_HID_DESCRIPTOR (9)
#define USB_WAIT_1000MS (1000)
#define SW_ACTIVE 0
#define SW R_PFS->PORT[0].PIN[8].PmnPFS_b.PIDR
#define SW_PDR R_PFS->PORT[0].PIN[8].PmnPFS_b.PDR
#define SW_PMR R_PFS->PORT[0].PIN[8].PmnPFS_b.PMR
static uint8_t g_buf[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* HID data */
static const uint8_t g_zero_data[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* zero data */
static uint16_t g_numlock = 0;
static uint8_t g_idle = 0;
uint8_t             g_remote_wakeup_enable = USB_OFF;
uint8_t             g_status               = NO_WRITING;
```

```
/***********************************************************************
 * Function Name : usb_cpu_getkeyno
 * Description : input key port
 * Arguments : none
 * Return value : uint16_t : key_no
 ***********************************************************************/
uint8_t usb_cpu_getkeyno (void)
{
    uint8_t key_buf = 0;
 if (SW_ACTIVE == SW)
    {
 if (sw_on_count[0] < SW_ON_THRESHOLD)
     {
           sw_on_count[0]++;
      }
    }
 else
    {
 if (sw_on_count[0] >= SW_ON_THRESHOLD)
     {
           key_buf |= SW_PUSH;
      }
      sw_on_count[0] = 0;
    }
 return key_buf;
} /* End of function usb_cpu_getkeyno() */
/***********************************************************************
 * Function Name : set_key_data
 * Description : Set key data to buffer
 * Arguments : none
 * Return value : none
 ***********************************************************************/
void set_key_data (uint8_t * p_buf)
{
```

```
 static uint8_t key_data;
    key_data = KBD_CODE_A;
    *(p_buf + 2) = key_data;
} /* End of function set_key_data() */
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/******************************************************************************
 * Function Name : usb_phid_example
 * Description : Peripheral HID application main process
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usb_phid_example (void)
{
    usb_event_info_t event_info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t event;
```

```
    uint8_t    * p_idle_value;
    uint8_t      sw_data;
    usb_info_t   info;
 fsp_err_t    ret_code = FSP_SUCCESS;
    uint8_t      send_data[16] BSP_ALIGN_VARIABLE(4);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
    memset(&event_info, 0, sizeof(usb_event_info_t));
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (FSP_SUCCESS == event_info.status)
      {
 if (DATA_WRITING == g_status)
      {
                        g_status = ZERO_WRITING;
                        g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
         }
 else
      {
                        g_status = DATA_WRITING;
```

```
                        usb_cpu_delay_xms(USB_WAIT_1000MS);
                        g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN,
USB_CLASS_PHID);
         }
         }
 break;
 case USB_STATUS_REQUEST
:                                                    /* Receive Class Request
*/
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_numlock, 2); /* Get the NumLock data (NumLock data is not used) */
         }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
      {
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
         }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                            send_data[i] = g_apl_configuration[18 + i];
         }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
         }
 else
```

```
     {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
        }
        }
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* Get SetIdle value */
                  p_idle_value = (uint8_t *) &event_info.setup.request_value;
                  g_idle = p_idle_value[1];
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
        }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
        }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                  g_status = DATA_WRITING;
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
        }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
        }
 else
     {
                  g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
```

```
        }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
                 p_idle_value = (uint8_t *) &event_info.setup.request_value;
                 g_idle = p_idle_value[1];
        }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* None */
        }
 else
     {
                 g_status = DATA_WRITING;
                 g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
        }
 break;
 case USB_STATUS_SUSPEND:
 break;
 case USB_STATUS_DETACH:
               g_remote_wakeup_enable = USB_OFF;
 break;
 default:
 break;
      }
      ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, 0);
 if (FSP_SUCCESS == ret_code)
     {
           sw_data = usb_cpu_getkeyno();
 if (USB_STATUS_SUSPEND == info.device_status)
     {
 if (0 != (sw_data & SW_PUSH))
```

```
      {
                    g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
          }
          }
       }
    }
} /* End of function usb_phid_example() */
```

# Descriptors

A template for PHID descriptors can be found in ra/fsp/src/r_usb_phid. Be sure to replace the vendor ID with your own.
Keyboard templates should be referred to r_usb_phid_descriptor_keyboard.c.template.
Mouse templates should be referred to r_usb_phid_descriptor_mouse.c.template.

### 5.2.6.44 USB PMSC (r_usb_pmsc)

Modules » Connectivity

This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface.

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

#### Detailed Description

# Overview

The r_usb_pmsc module combines with the r_usb_basic module to provide USB Peripheral It operates as a Mass Storage class driver (hereinafter referred to as PMSC).
The USB peripheral mass storage class driver (PMSC) comprises a USB mass storage class bulk-only transport (BOT) protocol.
When combined with a USB peripheral control driver and media driver, it enables communication with a USB host as a BOT-compatible storage device.

#### Features

The r_usb_pmsc module has the following key features:

- Storage command control using the BOT protocol

- Supports only SFF-8070i (ATAPI) Interface Sub-Class
- Response to mass storage device class requests from a USB host

# Configuration

### Build Time Configurations for r_usb_pmsc

The following build time configurations are defined in fsp_cfg/r_usb_pmsc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Vendor Information | Vendor Information must be 8 bytes long; pad with spaces if shorter. | Vendor | Specify the vendor information field (part of the Inquiry command response). |
| Product Information | Product Information must be 16 bytes long; pad with spaces if shorter. | Mass Storage | Specify the product information field (part of the Inquiry command response). |
| Product Revision Level | Product Revision Level must be 4 bytes long; pad with spaces if shorter. | 1.00 | Specify the product revision level field (part of the Inquiry command response). |
| Sector size | • 512<br>• 4096 | 512 | Specifies the sector size. |
| Number of Transfer Sectors | Please enter a number between 1 and 255. | 8 | Specify the maximum sector size to request with one data transfer. |

### Configurations for Connectivity > USB PMSC (r_usb_pmsc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PMSC (r_usb_pmsc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_pmsc0 | Module name. |

Refer to the USB (r_usb_basic) module for hardware configuration options.

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Class Requests

The class requests supported by this driver are shown below.

| Request | Code | Description |
|---|---|---|
| Bulk-Only Mass Storage Reset | 0xFF | Resets the connection interface to the mass storage device. |
| Get Max Logical Unit Number | 0xFE | Reports the logical numbers supported by the device. |

### Storage Commands

This driver supports the following storage commands.

| Command | Code | Description |
|---|---|---|
| TEST_UNIT_READY | 0x00 | Checks the state of the peripheral device. |
| REQUEST_SENSE | 0x03 | Gets the error information of the previous storage command execution result. |
| INQUIRY | 0x12 | Gets the parameter information of the logical unit. |
| READ_FORMAT_CAPACITY | 0x23 | Gets the formattable capacity. |
| READ_CAPACITY | 0x25 | Gets the capacity information of the logical unit. |
| READ10 | 0x28 | Reads data. |
| WRITE10 | 0x1A | Writes data. |
| MODE_SENSE10 | 0x5A | Gets the parameters of the logical unit. |

*Note*

*A STALL or FAIL error is sent to the host upon receipt of any command not listed in the above table.*

### BOT Protocol Overview

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out). The ATAPI storage commands and the response status are embedded in a Command Block Wrapper (CBW) and a Command Status Wrapper (CSW). The below image shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

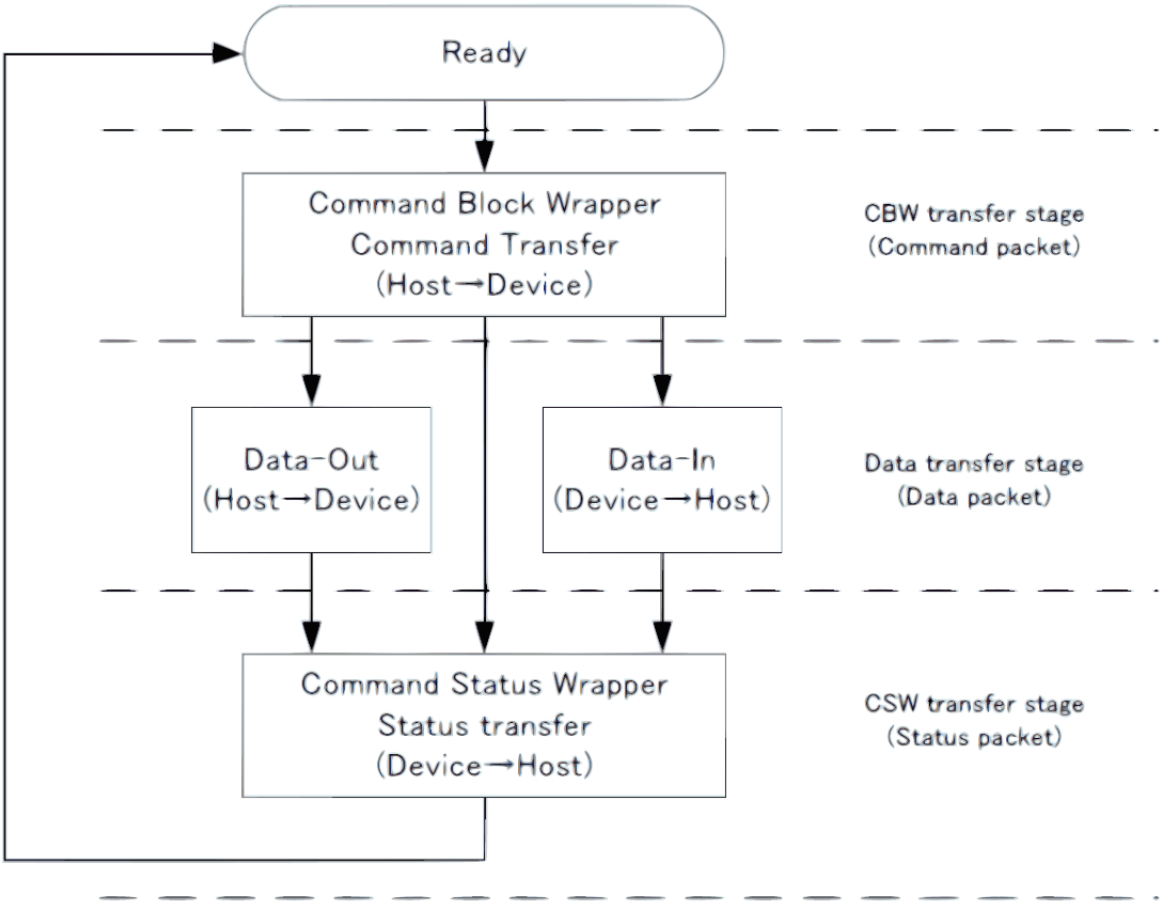



Figure 250: BOT protocol Overview

### Block Media Interface

PMSC implements a block media interface to enable access to higher-level modules. If the block media interface supports multiple media, users can select any media to access.

*Note*

*When the user develops the storage media driver, be sure to define the instance named "g_rm_block_media0".*

### Limitations

1. The driver always returns 0 in response to the GetMaxLun command.
2. The driver supports a sector size of 512 bytes only.
3. The only media currently supported by the block media interface is an SD card. The card must be inserted before initializing the driver.
4. When using DMA for Hi-Speed transfers continuous transfer mode must not be used in the USB Basic driver.
5. The storage area must be formatted before use.
6. When using the SD/MMC Block Media Implementation (rm_block_media_sdmmc), "Card Detection" must be set to "Not Used" in the SD/MMC Host Interface (r_sdhi) settings.
7. The driver does not support Low-speed.
8. This driver does not support simultaneous operation with USB Host device class.
9. Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

# Examples

### USB PMSC Example

In this example, when the evaluation board is connected to the host PC it is recognized as a removable disk and reading/writing files is possible. The FAT type is either FAT12, FAT16, or FAT32 depending on the size of the media used.

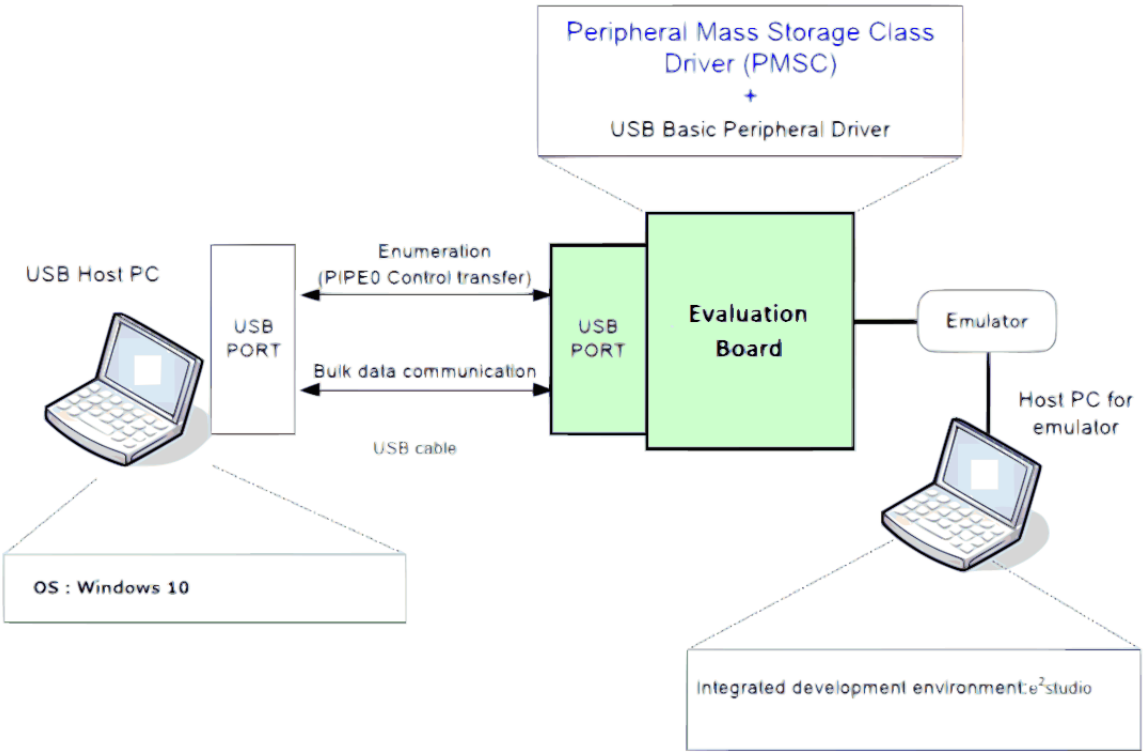



Figure 251: Example Operating Environment

```
#if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ******************************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
```

```
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
#endif /* (BSP_CFG_RTOS == 2) */
void usb_pmsc_example (void)
{
    usb_event_info_t usb_event;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#else
 usb_status_t event;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&usb_event, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
     {
 case USB_STATUS_CONFIGURED:
     {
 break;
           }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
     {
#if USB_SUPPORT_LPW == USB_APL_ENABLE
// @@ low_power_mcu();
```

```
#endif  /* USB_SUPPORT_LPW == USB_APL_ENABLE */
 break;
           }
 default:
      {
 break;
           }
       }
    }
} /* End of function usb_main() */
```

# Descriptor

A template for PMSC descriptors can be found in ra/fsp/src/r_usb_pmsc/r_usb_pmsc_descriptor.c.template. Also, please be sure to use your vendor ID.

### 5.2.6.45 USB PPRN (r_usb_pprn)

Modules » Connectivity

This module is USB Peripheral Printer Device Class Driver (PPRN). It implements the USB PPRN Interface.

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

#### Detailed Description

# Overview

The r_usb_pprn module combines with the r_usb_basic module to provide a USB Peripheral Printer Device Class (PPRN) driver. The PPRN driver conforms to the USB Printer Device class specifications and implements communication with a printer host.

#### Features

The r_usb_pprn module has the following functions:

- Data transfer to and from a USB host
- Response to the printer class requests

- Response to function references from the printer host

*Note*

*This driver is not guaranteed to provide USB printer operation in all scenarios. The developer must verify correct operation when connected to the targeted USB hosts.*

# Configuration

### Build Time Configurations for r_usb_pprn

The following build time configurations are defined in fsp_cfg/r_usb_pprn_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Connectivity > USB PPRN (r_usb_pprn)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PPRN (r_usb_pprn).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_pprn0 | Module name. |

### Clock Configuration

Refer to the USB (r_usb_basic) module.

### Pin Configuration

Refer to the USB (r_usb_basic) module.

# Usage Notes

### Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

| Request | Code | Description |
|---|---|---|
| GET_DEVICE_ID | 0x00 | Get IEEE 1284 Device ID String |
| GET_PORT_STATUS | 0x01 | Get Printer's Status |
| SOFT_RESET | 0x02 | Flushes all buffers and resets the Bulk OUT and Bulk IN pipes |

The data format of supported class requests is described below:

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_DEVICE_ID (0x00) | Config Index | Interface Alternate Setting | Maximum Length | 1284 Device ID String |

| | | | | | |
|---|---|---|---|---|---|
| 0xA1 | GET_PORT_STATUS (0x01) | 0 | Interface | 1 | Printer Status |
| 0x21 | SOFT_RESET (0x02) | 0 | Interface | 0 | None |

### Limitations

- This driver does not support USB Low-speed mode.
- This driver does not support simultaneous operation with USB Host device class.
- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

### USB PPRN Example

This is a minimal example for implementing PPRN in bare metal and FreeRTOS application.

```
/*******************************************************************************
 * Macro definitions
 ******************************************************************************/
/* Define device framework. */
#define DEMO_PROTOCOL (1U)  /* 1-Uni-dir, 2-Bi-dir */
#define VALUE_5BH (0x5BU)
/*******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
static uint8_t g_port_status = PORT_STATUS_BENIGN;
/* Device printer device ID. */
static uint8_t g_printer_device_id[] =
{
    0x00,                               /* data length */
    VALUE_5BH,
 'M',                                /* manufacturer (case sensitive) */
 'F',
 'G',
 ':',
 'G',
 'e',
 'n',
 'e',
 'r',
```

```
'i',
'c',
';',
'M',                                /* model (case sensitive) */
'D',
'L',
':',
'G',
'e',
'n',
'e',
'r',
'i',
'c',
'_',
'/',
'_',
'T',
'e',
'x',
't',
'_',
'O',
'n',
'l',
'y',
';',
'C',                                /* PDL command set */
'M',
'D',
':',
'1',
'2',
'8',
```

```
'4',
'.',
'4',
';',
'C',                            /* class */
'L',
'S',
':',
'P',
'R',
'I',
'N',
'T',
'E',
'R',
';',
'D',                            /* description */
'E',
'S',
':',
'G',
'e',
'n',
'e',
'r',
'i',
'c',
' ',
't',
'e',
'x',
't',
' ',
'o',
```

```
 'n',
 'l',
 'y',
 ' ',
 'p',
 'r',
 'i',
 'n',
 't',
 'e',
 'r',
 ';'
};
/***********************************************************************
 * Exported global functions (to be accessed by other files)
 ***********************************************************************/
void usb_pprn_example(void);
usb_instance_ctrl_t g_basic0_ctrl;
#if (BSP_CFG_RTOS == 2)
/***********************************************************************
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ***********************************************************************/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0));
} /* End of function usb_apl_callback */
```

```
#endif /* (BSP_CFG_RTOS == 2) */
/***********************************************************************************
 * Function Name : usb_pprn_example
 * Description : Peripheral Printer Class(PRN) application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************************/
void usb_pprn_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    usb_info_t       info;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    memset(&event_info, 0, sizeof(usb_event_info_t));
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif
 /* Handle the received event (if any) */
 switch (event)
     {
 case USB_STATUS_CONFIGURED:
 /* Initialization complete; get data from host */
               g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
 break;
```

```
 case USB_STATUS_WRITE_COMPLETE:
#if DEMO_PROTOCOL > 1
 if (FSP_SUCCESS == event_info.status)
      {
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
               }
#endif
 break;
 case USB_STATUS_READ_COMPLETE:
 if ((FSP_SUCCESS == event_info.status) || (FSP_ERR_USB_SIZE_SHORT ==
event_info.status))
      {
#if DEMO_PROTOCOL > 1
 /* Loop back to Host */
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PPRN);
#else
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PPRN);
#endif
               }
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PPRN_GET_DEVICE_ID == (event_info.setup.request_type & USB_BREQUEST))
      {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlDataSet(&g_basic0_ctrl, g_printer_device_id,
PRINTER_DEVICE_ID_LENGTH);
               }
 else if (USB_PRPN_GET_PORT_STATUS == (event_info.setup.request_type & USB_BREQUEST))
      {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlDataSet(&g_basic0_ctrl, &g_port_status, 1);
               }
 else if (USB_PPRN_SOFT_RESET == (event_info.setup.request_type & USB_BREQUEST))
```

```
     {
 R_USB_InfoGet(&g_basic0_ctrl, &info, event_info.device_address);
 R_USB_PeriControlStatusSet(&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
 /* [To do] Transport Abort */
                  g_usb_on_usb.stop(&g_basic0_ctrl, USB_TRANSFER_READ,
USB_CLASS_PPRN);
               }
 else
     {
               }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
      }
   }
}
```

# Descriptors

A template for PPRN descriptors can be found in ra/fsp/src/r_usb_pprn folder. Be sure to replace the vendor ID with your own.

### 5.2.6.46 USB Peripheral Vendor class (r_usb_pvnd)

Modules » Connectivity

#### Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

# Overview

USB Peripheral Vendor class works by combining r_usb_basic module.

# How to Configuration

The following shows FSP configuration procedure for USB Peripheral Vendor class.

- Select [New Stack]->[Middleware]->[USB]->[USB Peripheral Vendor class driver on r_usb_pvnd].

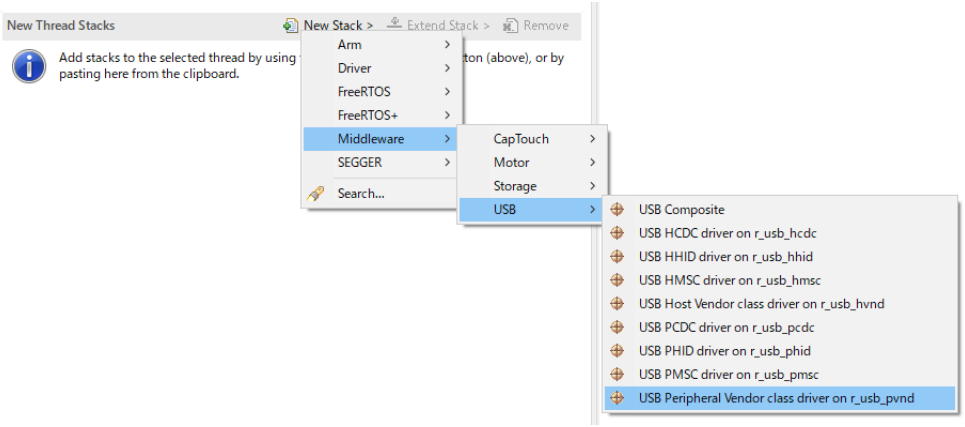



Figure 252: Select USB Peripheral Vendor Class

- The following is displayed when selecting [USB Peripheral Vendor class driver on r_usb_pvnd]. The user does not specify USB pipe number in Vendor class.

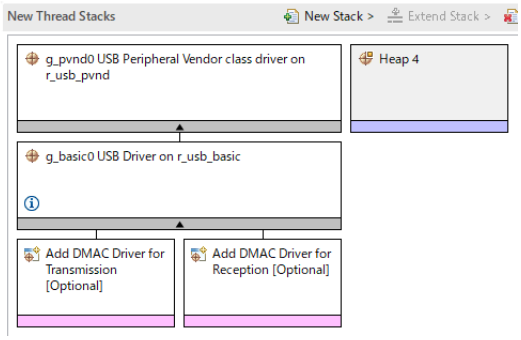



Figure 253: USB Peripheral Vendor Class Stack

# API

Use the following APIs in Peripheral Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()
  3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_PeriControlDataGet()
  2. R_USB_PeriControlDataSet()
  3. R_USB_PeriControlStatusSet()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

# USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.
FSP driver pipe configuration behaviors for the PVND Class, when it is used with the Composite PCDC+PVND Class is mentioned below.

- In Composite PCDC+PVND class, PCDC class interface is placed next to the PVND class interface in descriptor file. PVND Class Pipe configurations are as follows.
  - FSP driver(PVND) allocates Pipes in order from USB PIPE1 according to the order of the endpoint in descriptor(Bulk transfer).
  - FSP driver(PVND) allocates Pipes in order(decremental order) from USB PIPE9 according to the order of the endpoint in descriptor(Interrupt transfer).
  - PCDC and PVND must always use different USB PIPEs.

For example:

- In the Composite PCDC+PVND class, if there are 3 number of bulk transfers endpoints and one Interrupt transfer endpoint in descriptor file for PVND class and for this configuration FSP driver configures USB PIPE1, USB PIPE2 and USB PIPE3 for bulk data transfer and USB PIPE9 for Interrupt data transfer.
- Since hardware support total 5 number of Bulk transfer pipe configuration. So, in this scenario, for the PCDC class we have configurable USB PCDC Stack property to select available Pipes(USB PIPE 4 and USB PIPE 5 for Bulk and USB PIPE 6 for Interrupt transfer) from USB PCDC Stack configurator.

# Limitaions

- Please ignore the suspend or resume event occurs when attaching or detaching the USB cable.

# Descriptor

Template for Vendor class descriptor can be found in ra/fsp/src/r_usb_pvnd folder. Also, please be sure to use your vendor ID.

# Examples

This application program processes the following after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion
2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/***********************************************************************
**
 * Macro definitions
 ***********************************************************************
**/
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
/***********************************************************************
**
 * Function Name : usb_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_event_info_t *p_api_event : Control structure for USB API.
 * : usb_hdl_t cur_task : Task Handle
 * : uint8_t usb_state : USB_ON(USB_STATUS_REQUEST) / USB_OFF
 * Return value : none
 ***********************************************************************
**/
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    xQueueSend(g_apl_mbx_hdl, (const void *) &p_api_event, (TickType_t) (0))
} /* End of function usb_apl_callback() */
#endif /* (BSP_CFG_RTOS == 2) */
/***********************************************************************
```

```
**
 * Function Name : usb_pvnd_example
 * Description : Peripheral Vendor Class application main process
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
void usb_pvnd_example (void)
{
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type       = 0;
    uint8_t    pipe          = 0;
    uint8_t    is_zlp[2]      = {0, 0};
    uint32_t   request_length = 0;
    uint16_t   used_pipe      = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    memset(&event_info, 0, sizeof(usb_event_info_t));
 while (1)
    {
#if (BSP_CFG_RTOS == 2)  /* FreeRTOS */
       xQueueReceive(g_apl_mbx_hdl, (void *) &p_mess, portMAX_DELAY);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
```

```
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                buffer_init();
                is_zlp[0] = 0;
                is_zlp[1] = 0;
                g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
USB_CLASS_PVND);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if (0 != (used_pipe & (1 << pipe)))
      {
                      g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                           buf_type      = BUF_BULK;
                           bulk_out_pipe = pipe;
                        }
 else
      {
                           buf_type    = BUF_INT;
                           int_out_pipe = pipe;
                        }
                     }
 else
      {
 /* In Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
```

```
     {
                              buf_type    = BUF_BULK;
                              bulk_in_pipe = pipe;
                          }
else
     {
                              buf_type    = BUF_INT;
                              int_in_pipe = pipe;
                          }
                      }
                  }
              }
break;
          }
case USB_STATUS_READ_COMPLETE:
     {
if (FSP_ERR_USB_FAILED != event_info.status)
     {
if (bulk_out_pipe == event_info.pipe)
     {
                      buf_type = BUF_BULK;
                      pipe    = bulk_in_pipe;
                  }
else if (int_out_pipe == event_info.pipe)
     {
                      buf_type = BUF_INT;
                      pipe    = int_in_pipe;
                  }
else
     {
while (1)
     {
                          ;
                      }
```

```
                    }
                    buffer_check(buf_type, event_info.data_size);
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
                }
 break;
            }
 case USB_STATUS_WRITE_COMPLETE:
      {
 if (FSP_SUCCESS == event_info.status)
      {
 if (bulk_in_pipe == event_info.pipe)
      {
                        buf_type = BUF_BULK;
 if (1 == is_zlp[buf_type])
      {
                            pipe = bulk_out_pipe;
                        }
                    }
 else if (int_in_pipe == event_info.pipe)
      {
                        buf_type = BUF_INT;
 if (1 == is_zlp[buf_type])
      {
                            pipe = int_out_pipe;
                        }
                    }
 else
      {
 /* Nothing */
                    }
 if (1 == is_zlp[buf_type])
      {
                        is_zlp[buf_type] = 0;
```

```
                        buffer_clear(buf_type);
                        g_usb_on_usb.pipeRead(&g_basic0_ctrl,
&g_buf[buf_type][0], BUF_SIZE, pipe);
                    }
 else
     {
                        is_zlp[buf_type] = 1;
                        g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                    }
                }
 break;
            }
 case USB_STATUS_REQUEST:
     {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
     {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
     {
                    request_length = event_info.setup.request_length;
                    g_usb_on_usb.periControlDataGet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
     {
                    g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else
     {
```

```
 /* Nothing */
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                    g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_BULK][0],
BUF_SIZE, bulk_out_pipe);
                    g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_INT][0],
BUF_SIZE, int_out_pipe);
                }
 break;
            }
 case USB_STATUS_DETACH:
      {
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_pvnd_example() */
/***********************************************************************
**
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 ***********************************************************************
**/
```

```
static void buffer_init (void)
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
     {
           g_buf[j][i] = (uint8_t) i;
      }
    }
} /* End of function buffer_init() */
/***********************************************************************
**
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 ***********************************************************************
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
     {
 while (1)
     {
               ;
           }
      }
    }
} /* End of function buffer_check() */
```

```
/***********************************************************************
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 ***********************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
    }
} /* End of function buffer_clear() */
/***********************************************************************
**
 * End of function usb_mcu_init
 ***********************************************************************
**/
```

### 5.2.6.47 USB-TYPEC (r_usb_typec)

Modules » Connectivity

#### Functions

| | |
|---|---|
| fsp_err_t | R_USB_TYPEC_Open (usb_typec_ctrl_t *const p_ctrl, usb_typec_cfg_t const *const p_cfg)<br>Initialize USB Type-C module and USB Type-C driver. Implements usb_typec_api_t::open. More... |
| fsp_err_t | R_USB_TYPEC_Close (usb_typec_ctrl_t *const p_ctrl)<br>Uninitialize USB Type-C module and disable interrupts. Implements usb_typec_api_t::close. More... |

| | |
|---|---|
| fsp_err_t | R_USB_TYPEC_InfoGet (usb_typec_ctrl_t *const p_ctrl, usb_typec_info_t *p_info)<br>Get USB Type-C connect Information. Implements usb_typec_api_t::infoGet. More... |
| fsp_err_t | R_USB_TYPEC_LowPowerModeSet (usb_typec_ctrl_t *const p_ctrl, usb_typec_ccs_status_t status)<br>Set IRQ Mode for USB Type-C module by Low Power Mode. Implements usb_typec_api_t::lowPowerModeSet. More... |

### Detailed Description

Driver for the USB_TYPEC peripheral on RA MCUs. This module implements the USB TYPEC Interface.

# Overview

The r_usb_typec module is used in combination with r_usb_basic and other device class for USB communication and with the Type C module alone for Type-C charging.

### Features

The r_usb_typec module has the following key features:

- Detection of Source Connect/Disconnect (CC1/CC2 pin and USB_VBUS pin)
- Detection of Source Partner's Current Capability, Default USB, Type-C 1.5A, or Type-C 3.0A

# Usage Notes

### Basic Functions

The main functions of thie module are the following:

- Initialization and un-initialization of USB Type-C module.
- Getting of Type-C module information.

### Note

- Port Setting

If the user is using an MCU that supports the USB Type-C module and all of the following conditions are satisfied, please add code to control the port connected to External Rd immediately after calling the R_USB_TYPEC_Open function. (Because it is necessary to control External Rd after Internal Rd is turned ON.)

1. The USB operating mode is Device mode.
2. The power mode is BusPower. (External Rd is mounted on board)

```
R_USB_TYPEC_Open(&g_usbcc0_ctrl, &g_usbcc0_cfg);
R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_09, (uint32_t)
IOPORT_CFG_PORT_DIRECTION_OUTPUT | (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW);
```

- Debounce

Depending on the user board, the user may need code to debaunce the USB_VBUS pin.

# Examples

### USB Type-C Example

The following is example code for USB Type-C.

```
/*****************************************************************************
 * Function Name : usb_typec_apl_callback
 * Description : Callback function for Application program
 * Arguments : usb_typec_callback_args_t * p_typec_args : Callback function parameter
data.
 * Return value : none
 *****************************************************************************/
void usb_typec_apl_callback (usb_typec_callback_args_t * p_typec_args)
{
    g_typec_args = *p_typec_args;
 /* USB Type-C Callback Flag Set */
    g_typec_callback_flg = 1;
} /* End of function usb_apl_callback */
/*****************************************************************************
 * Function Name : usb_typec_example
 * Description : USB Type-C application main process
 * Arguments : none
 * Return value : none
 *****************************************************************************/
void usb_typec_example (void)
{
    usb_typec_info_t typec_info;
#if TYPEC_APL_SUPPORT_SOFTWARE_STANDBY
```

```
    g_lpm_on_lpm.open(&g_lpm_ctrl, &g_lpm_cfg);
#endif  /* TYPEC_APL_SUPPORT_SOFTWARE_STANDBY */
    g_typec_on_typec.open(&g_usbcc0_ctrl, &g_usbcc0_cfg);
 /* EK-RA2L2 ExRd1_En(P109) = L */
 R_IOPORT_PinCfg(&g_ioport_ctrl,
 BSP_IO_PORT_01_PIN_09,
                  (uint32_t) IOPORT_CFG_PORT_DIRECTION_OUTPUT |
                  (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW);
 /* EK-RA2L2 ExRd2_En(P110) = L */
 R_IOPORT_PinCfg(&g_ioport_ctrl,
 BSP_IO_PORT_01_PIN_10,
                  (uint32_t) IOPORT_CFG_PORT_DIRECTION_OUTPUT |
                  (uint32_t) IOPORT_CFG_PORT_OUTPUT_LOW);
 /* Waiting for USB Type-C event */
 while (1)
    {
 /* USB Type-C Callback Flag Check */
 if (1 == g_typec_callback_flg)
     {
 switch (g_typec_args.status)
     {
 /* Connection Status Change */
 case USB_TYPEC_STATUS_ISCN:
                  g_typec_on_typec.infoGet(&g_usbcc0_ctrl, &typec_info);
 switch (typec_info.connection_status)
     {
 case USB_TYPEC_CONNECTION_STATUS_UNATTACHED:
 /* None */
 break;
 case USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_DEFAULT:
 /* In the case of Power1.5.SNK or Power3.0.SNK, the PowerDefault event is notified
first. */
 /* None */
 break;
```

```
 case USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_1_5:
 /* None */
 break;
 case USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_3_0:
 /* None */
 break;
 default:
 /* None */
 break;
                 }
 break;
#if TYPEC_APL_SUPPORT_SOFTWARE_STANDBY
 /* VBUS Status Change */
 case USB_TYPEC_STATUS_ISVBUS:
 /* Note: Depending on the board, it may be necessary to use software to prevent VBUS
chattering.
  * */
 if (USB_TYPEC_VBUS_STATUS_OFF == g_typec_args.vbus_status)
     {
 /* USBCC CCS enable */
                      g_typec_on_typec.lowPowerModeSet(&g_usbcc0_ctrl,
USB_TYPEC_CCS_STATUS_ON);
 /* Software Standby */
                      g_lpm_on_lpm.lowPowerModeEnter(&g_lpm_ctrl);
 /* USBCC CCI enable */
                      g_typec_on_typec.lowPowerModeSet(&g_usbcc0_ctrl,
USB_TYPEC_CCS_STATUS_OFF);
                 }
 break;
#endif  /* TYPEC_APL_SUPPORT_SOFTWARE_STANDBY */
 default:
 /* None */
 break;
           }
```

```
 /* USB Type-C Callback Flag Clear */
            g_typec_callback_flg = 0;
       }
    }
} /* End of function usb_typec_example() */
```

## Data Structures

| | |
|---|---|
| struct | usb_typec_instance_ctrl_t |

## Data Structure Documentation

### ◆ usb_typec_instance_ctrl_t

| struct usb_typec_instance_ctrl_t | |
|---|---|
| USB TYPE-C control block. DO NOT INITIALIZE. Initialization occurs when usb_typec_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| usb_typec_cfg_t * | p_cfg |
| | Pointer to initial configurations. |
| | |
| uint32_t | module_number |
| | USB module number (USB_IP0/USB_IP1) |
| | |
| IRQn_Type | irq_typec_cci |
| | USBCC_CCI interrupt number storage variable. |
| | |
| IRQn_Type | irq_typec_ccs |
| | USBCC_CCS interrupt number storage variable. |
| | |
| uint32_t | ipl_typec_cci |
| | |

| | |
|---|---|
| | Variable to store the interrupt priority of USBCC_CCI. |
| | |
| uint32_t | ipl_typec_ccs |
| | Variable to store the interrupt priority of USBCC_CCS. |
| | |

### Function Documentation

#### ◆ R_USB_TYPEC_Open()

fsp_err_t R_USB_TYPEC_Open ( usb_typec_ctrl_t *const *p_ctrl*, usb_typec_cfg_t const *const *p_cfg* )

Initialize USB Type-C module and USB Type-C driver. Implements usb_typec_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_NOT_OPEN | USB TYPEC module is not open. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

#### ◆ R_USB_TYPEC_Close()

fsp_err_t R_USB_TYPEC_Close ( usb_typec_ctrl_t *const *p_ctrl*)

Uninitialize USB Type-C module and disable interrupts. Implements usb_typec_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_NOT_OPEN | USB TYPEC module is not open. |

### ◆ R_USB_TYPEC_InfoGet()

fsp_err_t R_USB_TYPEC_InfoGet ( usb_typec_ctrl_t *const *p_ctrl*, usb_typec_info_t * *p_info* )

Get USB Type-C connect Information. Implements usb_typec_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_NOT_OPEN | USB TYPEC module is not open. |

### ◆ R_USB_TYPEC_LowPowerModeSet()

fsp_err_t R_USB_TYPEC_LowPowerModeSet ( usb_typec_ctrl_t *const *p_ctrl*, usb_typec_ccs_status_t *status* )

Set IRQ Mode for USB Type-C module by Low Power Mode. Implements usb_typec_api_t::lowPowerModeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful completion. |
| FSP_ERR_NOT_OPEN | USB TYPEC module is not open. |
| FSP_ERR_USB_PARAMETER | Parameter error. |

## 5.2.6.48 USB_PCDC Communication Device (rm_comms_usb_pcdc)

Modules » Connectivity

### Functions

| | |
|---|---|
| fsp_err_t | RM_COMMS_USB_PCDC_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg)<br>Opens and configures the USB PCDC Comms module. Implements rm_comms_api_t::open. More... |
| fsp_err_t | RM_COMMS_USB_PCDC_Close (rm_comms_ctrl_t *const p_api_ctrl)<br>Disables specified USB PCDC Comms module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_USB_PCDC_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context) |

| | |
|---|---|
| | Updates the USB PCDC Comms callback. Implements rm_comms_api_t::callbackSet. More... |
| fsp_err_t | RM_COMMS_USB_PCDC_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)<br>Performs a read from the USB PCDC device. Implements rm_comms_api_t::read. More... |
| fsp_err_t | RM_COMMS_USB_PCDC_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write to the USB PCDC device. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_USB_PCDC_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs a write to, then a read from the USB device. Implements rm_comms_api_t::writeRead. More... |
| void | rm_comms_usb_pcdc_notify_application (rm_comms_usb_pcdc_instance_ctrl_t const *p_ctrl, rm_comms_event_t event) |
| void | rm_comms_usb_pcdc_callback_handler (usb_instance_ctrl_t *p_args, usb_hdl_t usb_handle, usb_onoff_t usb_onoff_status)<br>Common callback function called in the USB PCDC driver callback function. |

### Detailed Description

Middleware to implement a generic communications interface over USB_PCDC. This module implements the Communications Middleware Interface.

# Overview

The RM_COMMS_USB_PCDC module implements COMMS API for USB_PCDC interface.

### Features

The implementation of the USB_PCDC communications interfacehas the following key features:

- Non-blocking API for bare metal
- Non-blocking and blocking API for RTOS
- USB connection detection mode

# Configuration

### Build Time Configurations for rm_comms_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Connectivity > USB PCDC Communication Device (rm_comms_usb_pcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PCDC Communication Device (rm_comms_usb_pcdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| RTOS | | | |
| Recursive Write Mutex | • Do Not Use<br>• Use | Use | Lock device for writing in using RTOS. |
| Recursive Read Mutex | • Do Not Use<br>• Use | Use | Lock device for reading in using RTOS. |
| Mutex Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for recursive mutex operation in using RTOS. |
| Write Semaphore | • Do Not Use<br>• Use | Use | Block writing in using RTOS. |
| Read Semaphore | • Do Not Use<br>• Use | Use | Block reading in using RTOS. |
| Semaphore Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for semaphore operation in using RTOS. |
| Name | Name must be a valid C symbol | g_comms_usb_pcdc0 | Module name. |
| USB Connection Detection | • Disabled<br>• Baudrate Configuration<br>• Control Line State | Disabled | Enable it only when application wants to process the error of USB host tool not opened. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. |

# Usage Notes

USB connection detection has two modes: Baudrate Configuration mode and Control Line State mode. The former depends on line coding request, and the latter relies on control line state request. When it is enabled, rm_comms_usb_pcdc_write API will return FSP_ERR_NOT_ENABLED when USB host tool is not opened. By default it is disabled.

### Limitations

- RM_COMMS_API are not reentrant in non blocking mode
- When in blocking mode, RM_COMMS_USB_PCDC_Write() and RM_COMMS_USB_PCDC_Read() cannot be called in callback.
- RM_COMMS_USB_PCDC_WriteRead API is not implemented
- USB connection detection could be platform specific when it is enabled. For Windows USB host, Baudrate configuration mode should be used.

## Examples

### Basic Example

This is a basic example of minimal use of USB_PCDC communications implementation in an application.

```
void rm_comms_usb_pcdc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_COMMS_USB_PCDC_Open(&g_comms_usb_pcdc_ctrl, &g_comms_usb_pcdc_cfg);
 if (FSP_SUCCESS != err)
    {
 /* Handle any errors. */
    }
 while (true)
    {
 /* Send data. */
       g_err_flag = 0;
       g_tx_flag = 0;
       err       = RM_COMMS_USB_PCDC_Write(&g_comms_usb_pcdc_ctrl, g_tx_buf,
TX_BUF_LEN);
 if (FSP_SUCCESS != err)
     {
 /* Handle any errors. */
     }
 while ((0 == g_tx_flag) && (0 == g_err_flag))
     {
```

```
 /* Wait callback */
      }
 /* Receive data. */
       g_err_flag = 0;
       g_rx_flag = 0;
       err       = RM_COMMS_USB_PCDC_Read(&g_comms_usb_pcdc_ctrl, g_rx_buf,
RX_BUF_LEN);
 if (FSP_SUCCESS != err)
      {
 /* Handle any errors.*/
      }
 while ((0 == g_rx_flag) && (0 == g_err_flag))
      {
 /* Wait callback */
      }
    }
}
static void rm_comms_usb_pcdc_callback (rm_comms_callback_args_t * p_args)
{
 if (p_args->event == RM_COMMS_EVENT_TX_OPERATION_COMPLETE)
    {
       g_tx_flag = 1;
    }
 else if (p_args->event == RM_COMMS_EVENT_RX_OPERATION_COMPLETE)
    {
       g_rx_flag = 1;
    }
 else
    {
       g_err_flag = 1;
    }
}
```

# Descriptor

A template for PCDC descriptors can be found in ra/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

### Data Structures

| | |
|---|---|
| struct | rm_comms_usb_pcdc_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_comms_usb_pcdc_instance_ctrl_t

| struct rm_comms_usb_pcdc_instance_ctrl_t | |
|---|---|
| Communications middleware control structure. | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_comms_cfg_t const * | p_cfg |
| | Middleware configuration. |
| | |
| rm_comms_usb_pcdc_extended_cfg_t const * | p_extend |
| | Pointer to extended configuration structure. |
| | |
| usb_callback_args_t * | p_usb_args |
| | Pointer to usb callback args. |
| | |
| usb_pcdc_ctrllinestate_t | ctrl_line_state |
| | Line State Control. |
| | |
| usb_pcdc_linecoding_t | line_coding |
| | Line Coding to store COM port settings. |
| | |
| void(* | p_callback )(rm_comms_callback_args_t *p_args) |
| | Pointer to callback that is called when a usb_status_t occurs. |

| | |
|---|---|
| void * | p_context |
| | Pointer to context passed into callback function. |

### Function Documentation

#### ◆ RM_COMMS_USB_PCDC_Open()

fsp_err_t RM_COMMS_USB_PCDC_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the USB PCDC Comms module. Implements rm_comms_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | USB PCDC Comms module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

#### ◆ RM_COMMS_USB_PCDC_Close()

fsp_err_t RM_COMMS_USB_PCDC_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified USB PCDC Comms module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_USB_PCDC_CallbackSet()

fsp_err_t RM_COMMS_USB_PCDC_CallbackSet ( rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context* )

Updates the USB PCDC Comms callback. Implements rm_comms_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_USB_PCDC_Read()

fsp_err_t RM_COMMS_USB_PCDC_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the USB PCDC device. Implements rm_comms_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_ENABLED | USB device needs connecting from USB host to operate. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_USB_PCDC_Write()

fsp_err_t RM_COMMS_USB_PCDC_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the USB PCDC device. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_ENABLED | USB device needs connecting from USB host to operate. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_USB_PCDC_WriteRead()

fsp_err_t RM_COMMS_USB_PCDC_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs a write to, then a read from the USB device. Implements rm_comms_api_t::writeRead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Not supported. |

### ◆ rm_comms_usb_pcdc_notify_application()

void rm_comms_usb_pcdc_notify_application ( rm_comms_usb_pcdc_instance_ctrl_t const * *p_ctrl*, rm_comms_event_t *event* )

(end addtogroup RM_COMMS_USB_PCDC)

## 5.2.7 DSP

Modules

### Detailed Description

DSP Modules.

**Modules**

| | |
|---|---|
| | CMSIS DSP H/W Acceleration (rm_cmsis_dsp)<br>Middleware to implement Arm CMSIS DSP by using 32-bit Multiply-Accumulator (MACL). |
| | IIR Filter Accelerator (r_iirfa)<br>Driver for the IIRFA peripheral on RA MCUs. This module implements the IIR Interface. |

### 5.2.7.1 CMSIS DSP H/W Acceleration (rm_cmsis_dsp)

Modules » DSP

Middleware to implement Arm CMSIS DSP by using 32-bit Multiply-Accumulator (MACL).

# Overview

The 32-bit Multiply-Accumulator (MACL) provides hardware acceleration for calculation of CMSIS DSP functions. After initializing this module, refer to the Arm documentation to use the file system: https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html

**Features**

The MACL module supports the following features:

| No | MACL API | Feature |
|---|---|---|
| 1 | arm_mult_q31 | Q31 vector multiplication |
| 2 | arm_scale_q31 | Multiplies a Q31 vector by a scalar |
| 3 | arm_mat_mult_q31 | Q31 matrix multiplication |
| 4 | arm_mat_vec_mult_q31 | Q31 matrix and vector multiplication |
| 5 | arm_mat_scale_q31 | Q31 matrix scaling |
| 6 | arm_biquad_cascade_df1_q31 | Processing function for the Q31 Biquad cascade filter |
| 7 | arm_conv_partial_q31 | Partial convolution of Q31 sequences |
| 8 | arm_conv_q31 | Convolution of Q31 sequences |
| 9 | arm_correlate_q31 | Correlation of Q31 sequences |

| | | |
|---|---|---|
| 10 | arm_fir_decimate_q31 | Processing function for the Q31 FIR decimator |
| 11 | arm_fir_interpolate_q31 | Processing function for the Q31 FIR interpolator |
| 12 | arm_fir_q31 | Processing function for Q31 FIR filter |
| 13 | arm_fir_sparse_q31 | Processing function for the Q31 sparse FIR filter |
| 14 | arm_lms_norm_q31 | Processing function for Q31 normalized LMS filter |
| 15 | arm_lms_q31 | Processing function for Q31 LMS filter |

*Note*

*The MACL hardware acceleration when executing 5 APIs including arm_scale_q31, arm_mat_scale_q31, arm_correlate_q31, arm_fir_q31, and arm_lms_q31 with 1-element input on IAR compiler will not be better than ARM CMSIS software. The MACL hardware is useful when executing with larger amounts of input.*

# Configuration

### CMSIS DSP Acceleration Configuration

To enable hardware acceleration for Arm CMSIS DSP, the stack "Arm CMSIS6 DSP Acceleration" of CMSIS DSP must be filled with MACL (rm_cmsis_dsp). Otherwise the CMSIS DSP functions will perform the calculation by using software. This can be done at the Stack Configuration.

# Usage Notes

### Hardware Support

MACL module is required for using this middleware. Refer to the MCU hardware user's manual or datasheet to determine if it has MACL support.

# Examples

### Basic Example

About examples, how to use the CMSIS DSP APIs refer to the link: https://github.com/ARM-software/CMSIS-DSP/tree/main/Examples/ARM.

### 5.2.7.2 IIR Filter Accelerator (r_iirfa)

Modules » DSP

**Functions**

| | |
|---|---|
| fsp_err_t | R_IIRFA_Open (iir_ctrl_t *const p_api_ctrl, iir_cfg_t const *const p_cfg) |

| | |
|---|---|
| fsp_err_t | R_IIRFA_Filter (iir_ctrl_t *const p_api_ctrl, float const *p_data_in, float *p_data_out, uint16_t const num_samples) |
| fsp_err_t | R_IIRFA_Configure (iir_ctrl_t *const p_api_ctrl, iir_filter_cfg_t const *const p_filter_cfg) |
| fsp_err_t | R_IIRFA_StatusGet (iir_ctrl_t *const p_api_ctrl, iir_status_t *const p_status) |
| fsp_err_t | R_IIRFA_Close (iir_ctrl_t *const p_api_ctrl) |
| __STATIC_INLINE float | R_IIRFA_SingleFilter (iir_ctrl_t *const p_api_ctrl, float data_in) |

### Detailed Description

Driver for the IIRFA peripheral on RA MCUs. This module implements the IIR Interface.

# Overview

The IIR Filter Accelerator (IIRFA) provides hardware acceleration for calculation of single-precision floating point direct form 2 biquad IIR filters. Up to 32 biquad stages can be configured at a time.

### Features

The IIRFA module supports the following features:

- Up to 16 different concurrent configurations/channels
- 32 biquad filter stages (shared between all channels)
- Runtime filter configuration
- ECC error detection for coefficient and delay data
    - 1-bit error correction, 2-bit error detection

# Configuration

### Build Time Configurations for r_iirfa

The following build time configurations are defined in fsp_cfg/r_iirfa_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Polling Mode | • Enabled<br>• Disabled | Enabled | When enabled the IIRFA driver will poll for completion before reading the output register. This prevents |

| | | | |
|---|---|---|---|
| | | | IIRFA operations from delaying global interrupt processing at the cost of slower filter performance. |
| Software Loop Unroll Depth | Refer to the RA Configuration tool for available options. | 1 Sample | Select the number of samples to process for every loop. This setting generally only affects filters that use 1 biquad stage. |
| ECC Support | • Disabled<br>• Enabled<br>• Enabled (no writeback) | Enabled | Select whether to detect ECC errors. When set to 'Enabled (no writeback)' 1-bit ECC errors will not be corrected. |
| Rounding Mode | • Nearest<br>• Toward zero | Nearest | Select how to round calculation results. |

**Configurations for DSP > IIR Filter Accelerator (r_iirfa)**

This module can be added to the Stacks tab via New Stack > DSP > IIR Filter Accelerator (r_iirfa). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_iirfa0 | Module name. |
| Channel | Value must be an integer between 0 and 15 | 0 | Select the IIRFA channel. |

**Clock Configuration**

The clock source for the IIRFA peripheral is ICLK.

**Pin Configuration**

IIRFA does not have any pin connections.

**Filter Configuration**

Before using IIRFA to filter data the coefficients and state variables must be initialized. See the Filter Configuration example for how to initialize an iir_filter_cfg_t struct to pass to R_IIRFA_Configure.

*Note*

*There are 32 total biquad stages in the peripheral shared across all channels. Channels may not select the same stages, so it is important to choose iir_filter_cfg_t::stage_base and iir_filter_cfg_t::stage_num carefully to ensure no overlap.*

# Usage Notes

### Maximizing Performance

The optimum configuration for IIRFA is dependent on the application. It is recommended to consider the following in regards to your project to determine what settings may be ideal.

Regardless of configuration, each filter stage takes 2 ICLK cycles to process per sample and an additional 5 cycles to write state values back to registers. This means single-sample operations only take 2 cycles per stage while multi-sample operations take 7. Additional overhead cycles are required to load and store each sample. The following suggestions may improve performance:

- Either process more data at a time with R_IIRFA_Filter or only one sample at a time using R_IIRFA_SingleFilter
- Use the unrolled software loop option
    - Provide data in a multiple of the unroll depth
- Disable polling (see warning)

### Polling Mode

By default, the driver will poll a status flag for completion after writing input data to IIRFA. Disabling polling significantly improves performance when using a low number of stages, but may cause higher global interrupt latency during processing.

*Note*

*If only one stage is used it is recommended to disable polling as IIRFA will typically execute faster than code resulting in no wait cycles. If polling must be used with a one stage filter, please note that in some situations the Arm CMSIS DSP Library functions arm_biquad_cascade_df2T_init_f32 and arm_biquad_cascade_df2T_f32 may provide better performance. It up to the user to evaluate performance within their own project.*

Warning

When polling is disabled, filter operation works by writing a value to IIRCHnINP then immediately reading IIRCHnOUT. While this maximizes performance, the core will wait for output data to become available before continuing execution. **While execution waits for IIRCHnOUT no interrupts will be processed by the core.** The maximum wait time for a 32-stage filter may be up to approximately 64 ICLK cycles for single sample processing or 224 ICLK cycles for multi-sample processing (decreasing linearly with the number of stages used).

### Single Sample Processing

In applications such as motor control where each sample needs immediate processing, the inline function R_IIRFA_SingleFilter is provided. This function has no parameter checking, takes one sample, and returns a filtered sample. Polling is supported by this function (if configured).

### ECC Errors

If configured, R_IIRFA_Filter will return an error if a 1- or 2-bit ECC error has occurred.

1-bit errors are automatically corrected unless writeback is disabled. 2-bit errors cannot be automatically corrected. Reset the coefficient and filter data by calling R_IIRFA_Configure if a non-correctable ECC error is reported.

### Limitations

- A total of 32 stages may be configured at any one time across all channels.
- When polling is disabled core execution is halted while waiting for data to become available. (See warning)

# Examples

### Filter Configuration

Below is an example of how to initialize a filter configuration to pass to R_IIRFA_Configure.

```
#define FILTER_STAGE_NUM (2)
/* Biquad coefficients (4th order Butterworth 200Hz lowpass on 44.1KHz input) */
iir_filter_coeffs_t gp_iirfa0_filter_coeffs[FILTER_STAGE_NUM] =
{
    {
       .b0 = 1.000000000F,
       .b1 = 2.000000000F,
       .b2 = 1.000000000F,
       .a1 = -1.947914029F,
       .a2 = 0.948705125F,
    },
    {
       .b0 = 1.000000000F,
       .b1 = 2.000000000F,
       .b2 = 1.000000000F,
       .a1 = -1.977625722F,
       .a2 = 0.978428884F,
    },
};
/* Biquad state data (clear to start) */
iir_filter_state_t gp_iirfa0_filter_state[FILTER_STAGE_NUM] = {0};
/* Filter configuration */
iir_filter_cfg_t g_iirfa0_filter_cfg =
{
    .p_filter_coeffs = gp_iirfa0_filter_coeffs, // Pointer to filter coefficient
array
```

```
    .p_filter_state = gp_iirfa0_filter_state, // Pointer to filter state data array
    .stage_base      = 0,                     // Which hardware biquad stage to
start allocation from (0-31)
    .stage_num       = 2,                     // Number of stages to allocate
};
```

### Software Example

The following is a basic example of configuring and using a filter with IIRFA.

```
#define NUM_SAMPLES (128)
#define TWO_PI (2.0F * (float) M_PI)
float gp_data_in[NUM_SAMPLES];
float gp_data_out[NUM_SAMPLES];
void iirfa_filter_example (void)
{
 fsp_err_t err;
 /* Initialize the IIRFA module */
    err = R_IIRFA_Open(&g_iirfa0_ctrl, &g_iirfa0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize filter coefficients and state variables */
    err = R_IIRFA_Configure(&g_iirfa0_ctrl, &g_iirfa0_filter_cfg);
    assert(FSP_SUCCESS == err);
 /* Get input data to be filtered */
    get_input_data(gp_data_in);
 /* Filter data */
    err = R_IIRFA_Filter(&g_iirfa0_ctrl, gp_data_in, gp_data_out, NUM_SAMPLES);
 /* R_IIRFA_Filter will return FSP_ERR_INVALID_RESULT when one or more calculations
results in infinity. */
 if (FSP_ERR_INVALID_RESULT == err)
    {
 /* Handle error */
    }
 else
```

```
    {
      assert(FSP_SUCCESS == err);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | iirfa_instance_ctrl_t |

## Data Structure Documentation

### ◆ iirfa_instance_ctrl_t

| struct iirfa_instance_ctrl_t |
|---|
| IIRFA instance control block. |

## Function Documentation

### ◆ R_IIRFA_Open()

| fsp_err_t R_IIRFA_Open ( iir_ctrl_t *const *p_api_ctrl*, iir_cfg_t const *const *p_cfg* ) |
|---|
| Prepare an IIR channel for filtering.<br><br>**Return values**<br>FSP_SUCCESS — The channel was successfully opened.<br>FSP_ERR_ASSERTION — One or both of the parameters was NULL.<br>FSP_ERR_IP_CHANNEL_NOT_PRESENT — An invalid channel was selected.<br>FSP_ERR_ALREADY_OPEN — The instance is already opened. |

### ◆ R_IIRFA_Filter()

fsp_err_t R_IIRFA_Filter ( iir_ctrl_t *const *p_api_ctrl*, float const * *p_data_in*, float * *p_data_out*, uint16_t const *num_samples* )

Start a filter operation on the specified data.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data has been successfully filtered. |
| FSP_ERR_ASSERTION | One of the provided pointers is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_INVALID_ARGUMENT | num_samples is zero. |
| FSP_ERR_INVALID_RESULT | The result of one or more calculations was +/- infinity. |
| FSP_ERR_NOT_INITIALIZED | The filter is not configured. |
| FSP_ERR_IIRFA_ECC_1BIT | A 1-bit ECC error was detected. |
| FSP_ERR_IIRFA_ECC_2BIT | A 2-bit ECC error was detected. |

### ◆ R_IIRFA_Configure()

fsp_err_t R_IIRFA_Configure ( iir_ctrl_t *const *p_api_ctrl*, iir_filter_cfg_t const *const *p_filter_cfg* )

Configure filter coefficients and state values.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration successful. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_INVALID_ARGUMENT | Invalid filter stage selection. |
| FSP_ERR_IN_USE | One or more requested filter stages are currently in use. |

### ◆ R_IIRFA_StatusGet()

fsp_err_t R_IIRFA_StatusGet ( iir_ctrl_t *const *p_api_ctrl*, iir_status_t *const *p_status* )

Read the current filter state variables.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information read successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |

### ◆ R_IIRFA_Close()

fsp_err_t R_IIRFA_Close ( iir_ctrl_t *const *p_api_ctrl*)

Stop filter operations and close the channel instance.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |

### ◆ R_IIRFA_SingleFilter()

__STATIC_INLINE float R_IIRFA_SingleFilter ( iir_ctrl_t *const *p_api_ctrl*, float *data_in* )

Perform a single inline filter operation.

***Note***

*This function is intended to be used in performance-critical applications. As such, no parameter checking or error validation is provided.*

## 5.2.8 Graphics

Modules

### Detailed Description

Graphics Modules.

### Modules

| | |
|---|---|
| | Azure RTOS GUIX Port (rm_guix_port) |
| | Capture Engine Unit (r_ceu)<br>Driver for the CEU peripheral on RA MCUs. This module implements the CAPTURE Interface. |
| | D/AVE 2D Port Interface (r_drw)<br>Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D. |
| | Display Serial Interface (r_vin)<br>Driver for the VIN peripheral on RA MCUs. This module implements the CAPTURE Interface. |
| | Graphics LCD (r_glcdc)<br>Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface. |
| | JPEG Codec (r_jpeg)<br>Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface. |
| | LVGL Port (rm_lvgl_port)<br>LVGL graphics library port for RA MCUs. |
| | MIPI Display Serial Interface (r_mipi_dsi)<br>Driver for the MIPI DSI peripheral on RA MCUs. This module implements the Display Interface. |
| | Parallel Data Capture (r_pdc)<br>Driver for the PDC peripheral on RA MCUs. This module implements the CAPTURE Interface. |
| | SEGGER emWin RA Port (rm_emwin_port)<br>SEGGER emWin port for RA MCUs. |
| | Segment LCD (r_slcdc) |

| | |
|---|---|
| | Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface. |

### 5.2.8.1 Azure RTOS GUIX Port (rm_guix_port)

Modules » Graphics

#### Functions

| | |
|---|---|
| UINT | rm_guix_port_hw_initialize (GX_DISPLAY *p_display) |

#### Detailed Description

# Overview

The Azure RTOS GUIX Port module provides the configuration and hardware acceleration support necessary for use of GUIX on RA products. The port provides full integration with the graphics peripherals (GLCDC, DRW and JPEG).

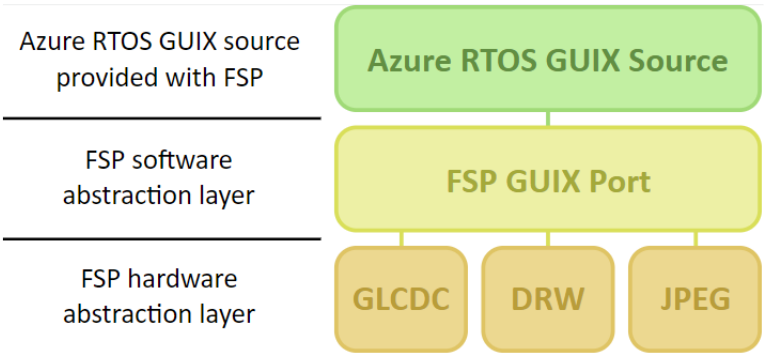



Figure 254: Azure RTOS GUIX Port Block Diagram

*Note*

*This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. For information about how to use GUIX and GUIX Studio (including example code) please consult the Azure RTOS GUIX documentation.*

### Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- DRW Engine (D/AVE 2D Port Interface (r_drw))
  - Drawing bitmaps
  - 8, 4 and 1bpp uncompressed and compressed (RLE) font rendering
  - Line and shape drawing
  - Anti-aliased operations
    - Circle stroke and fill
    - Polygon stroke and fill

        - Lines and arcs
- JPEG Codec (r_jpeg)
    - JPEG decoding
- Graphics LCD (r_glcdc)
    - Brightness, contrast and gamma correction
    - Pixel format conversion (framebuffer to LCD)

# Configuration

### Build Time Configurations for gx

The following build time configurations are defined in fsp_cfg/azure/gx/gx_user.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Hardware Acceleration | | | |
| JPEG Codec Support | MCU Specific Options | | Select whether or not to use the JPEG Codec for hardware acceleration. |
| DRW Engine Support | • Enabled<br>• Disabled | Enabled | Select whether or not to use the DRW Engine for hardware acceleration. |
| Max DRW Operations | Value must be a positive integer | 30 | Specifies the maximum number of DRW operations before flushing the display list. Reducing this value may reduce the peak heap used by the application but may reduce performance. |
| Internal Thread | | | |
| Stack Size | Value must be greater than zero | 4096 | GUIX internal thread stack size in bytes. Must be greater than zero. |
| Priority | Value must be between 0 to 31 | 30 | Priority of the GUIX Internal Thread. The value must be between 0 to 31. |
| Time Slice | Value must be in range 0 to 0xFFFFFFFF | 10 | Time Slice value of the GUIX Internal Thread. The value must be between 0 (TX_NO_TIME_SLICE) to 0xFFFFFFFF. |
| System Timer (ms) | Value must be greater | 20 | GUIX system timer |

| | | | |
|---|---|---|---|
| | than or equal to 10 | | period (GX_SYSTEM_TIMER_MS). This value will be internally converted to RTOS ticks and will be rounded down to the next smallest multiple of the RTOS tick period (1000 / TX_TIMER_TICKS_PER_SECOND). |
| Multithread Support | • Disabled<br>• Enabled | Enabled | Must be enabled if GUIX functions are called from multiple threads. Set to Disabled when calling GUIX from only one thread to reduce system overhead. |
| UTF8 Support | • Disabled<br>• Enabled | Enabled | Select whether to enable or disable support for UTF8 characters. |
| Event Queue Size | Value must be greater than zero | 48 | Maximum number of events in the GUIX event queue. |
| Enable GX_WIDGET User Data | • Enabled<br>• Disabled | Disabled | Set to Enabled to to use the gx_widget_user_data member in the GX_WIDGET structure. |

**Build Time Configurations for rm_guix_port**

The following build time configurations are defined in fsp_cfg/middleware/rm_guix_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| DRW Buffer Cache | • Enabled<br>• Disabled | Enabled | Enabling this option significantly improves DRW Engine performance. Set to Disabled only if Display underflow events are triggered under high graphics load. |

**Configurations for Graphics > Azure RTOS GUIX Port (rm_guix_port)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Display Rotation | | | |
| Screen Orientation | • None | None | Select the display |

| | | | |
|---|---|---|---|
| | • CW (90 degrees)<br>• FLIP (180 degrees)<br>• CCW (270 degrees) | | orientation specified in the GUIX Studio project. The Canvas Buffer must be enabled when rotating 180 degrees (FLIP). |
| Use Canvas Buffer | • Enabled<br>• Disabled | Disabled | When screen rotation is set to 180 degrees (FLIP), a canvas buffer must be used. The canvas buffer size will be the same as a frame buffer for the display module. |
| Canvas Buffer Memory Section | This property must be a valid section name | bss | Specify the memory section where the GUIX Canvas Buffer will be allocated. |
| JPEG Decoding | | | |
| Work Buffer Size | Must be a positive integer greater than 0 | 0xC800 | Specify the JPEG work buffer size in bytes. A larger buffer can reduce JPEG decode/draw times. The buffer will not be allocated if JPEG Codec support is disabled.<br><br>Unless you are sure of the subsampling used in and the size of the input JPEG images it is recommended to allocate at least 16 framebuffer lines of memory. |
| Buffer Memory Section | This property must be a valid section name | bss | Specify the memory section where the JPEG Work Buffer will be allocated. |
| Name | Name must be a valid C symbol | g_rm_guix_port0 | Module name. |
| Target Display Layer | • Graphics Layer 1<br>• Graphics Layer 2 | Graphics Layer 1 | Specify which graphics screen to inherit the buffer and display dimensions from. |
| Callback Function | Must be a valid C symbol | NULL | If a callback function is provided it will be called when Display events occur. |

### Hardware Configuration

No clocks or pins are directly required by this module. Please consult the submodules' documentation for their requirements.

## Usage Notes

### Getting Started

To get started with GUIX in an RA project the following must be performed:

1. In e² studio, open the RA Configuration editor for your GUIX project
2. Select or create a thread
3. Add GUIX to your project in the Stacks view by clicking **New Stack -> Azure RTOS -> GUIX**
4. Ensure the configuration options for GUIX and the port layer are set as necessary for your application
5. Set the properties for the GLCDC module to match the timing and memory requirements of your display panel
6. Set the input color format in the GLCDC module (Input -> Graphics Layer * -> General -> Color format) and the output color format in the JPEG Codec module if applicable (Decode -> Output color format) per your project specification
7. Click the BSP tab in the configuration editor and confirm the heap size in the Properties pane is sufficient (see Note below)
8. Click Generate Project Content to save and commit configuration changes
9. Drop the Quick Setup entry in Developer Assistance into the desired thread entry C file and update the items marked with TODO as necessary
10. Call the Quick Setup function from the thread entry function (or where desired)

At this point the project is now ready to build and run your GUIX Studio project. Please refer to the documentation for Azure RTOS GUIX and GUIX Studio for details on how to create and edit a GUI application.

*Note*

*It is recommended to start with 8K-32K of heap to begin development. Actual heap use is typically far lower than this but must be characterized by the developer.*

### Using Hardware Acceleration

In most cases there is no need to perform additional configuration to ensure the DRW Engine is used. However, there are some guidelines that should be followed:

- Avoid transparent pixelmaps in 8-bit display mode as they are rendered in software. In particular, ensure PNGs to be used in 8bpp GUIX Studio projects are saved without transparency data if no transparency is needed.
- The following items may require a large heap to draw successfully:
  - Polygons (more sides = more heap)
  - Filled arcs and ellipses (more framebuffer lines occupied = more heap)
  - gx_canvas_pixelmap_tile (more tiles = more heap)
- When using hardware acceleration, images used for tile fill of shapes must have dimensions that are a power of 2. This limitation does not apply to gx_canvas_pixelmap_tile as well as certain arc/ellipse fill functions as GUIX manually draws pixelmaps to fill these shapes (at the expense of heap space).

# Examples

### Basic Example

This is a basic example demonstrating how to get GUIX up and running given an existing GUIX Studio project. A template for this code is available in Developer Assistance for the GUIX Port module.

*Note*

*GUIX manages the GLCDC, DRW and JPEG Codec submodules internally; they do not need to be opened directly.*

```
GX_WINDOW_ROOT * p_window_root;
void guix_user_start (void)
{
 /* Initialize GUIX */
    gx_system_initialize();
 /* Configure GUIX Studio project main display and get a pointer to the root window
*/
    gx_studio_display_configure(MAIN_DISPLAY,
 rm_guix_port_hw_initialize,
                                MAIN_DISPLAY_LANGUAGE_ENGLISH,
                                MAIN_DISPLAY_THEME,
                                &p_window_root);
 /* Set pointer to the first buffer generated by the configuration
(rm_guix_port_canvas) */
    gx_canvas_memory_define(p_window_root->gx_window_root_canvas,
                            rm_guix_port_canvas,

p_window_root->gx_window_root_canvas->gx_canvas_memory_size);
 /* Create and show the root window in the GUIX Studio project */
    gx_studio_named_widget_create("root_widget_name", (GX_WIDGET *) p_window_root,
GX_NULL);
    gx_widget_show(p_window_root);
 /* Start GUIX */
    gx_system_start();
 /* GUIX will continue to run in its own thread */
}
```

## Data Structures

| | |
|---|---|
| struct | rm_guix_port_callback_args_t |

## Enumerations

| | |
|---|---|
| enum | rm_guix_port_device_t |
| enum | rm_guix_port_event_t |

## Data Structure Documentation

### ◆ rm_guix_port_callback_args_t

| struct rm_guix_port_callback_args_t | | |
|---|---|---|
| Callback arguments for the FSP GUIX Port | | |
| Data Fields | | |
| rm_guix_port_device_t | device | Device code. |
| rm_guix_port_event_t | event | Event code of the low level hardware. |
| uint32_t | error | Error code if RM_GUIX_PORT_EVENT_ERROR. |

## Enumeration Type Documentation

### ◆ rm_guix_port_device_t

| enum rm_guix_port_device_t | |
|---|---|
| Low level device code for the GUIX | |
| Enumerator | |
| RM_GUIX_PORT_DEVICE_NONE | Non hardware. |
| RM_GUIX_PORT_DEVICE_DISPLAY | Display device. |
| RM_GUIX_PORT_DEVICE_DRW | 2D Graphics Engine |
| RM_GUIX_PORT_DEVICE_JPEG | JPEG Codec. |

### ◆ rm_guix_port_event_t

| enum rm_guix_port_event_t | |
|---|---|
| Display event codes | |
| Enumerator | |
| RM_GUIX_PORT_EVENT_ERROR | Low level driver error occurs. |
| RM_GUIX_PORT_EVENT_DISPLAY_VSYNC | Display interface VSYNC. |
| RM_GUIX_PORT_EVENT_UNDERFLOW | Display interface underflow. |

## Function Documentation

### ◆ rm_guix_port_hw_initialize()

UINT rm_guix_port_hw_initialize ( GX_DISPLAY * *p_display*)

Callback function to be passed to gx_studio_display_configure in order to start hardware modules.

Example:

```
 /* Configure GUIX Studio project main display and get a pointer to the root window
*/
    gx_studio_display_configure(MAIN_DISPLAY,
 rm_guix_port_hw_initialize,
                                MAIN_DISPLAY_LANGUAGE_ENGLISH,
                                MAIN_DISPLAY_THEME,
                                &p_window_root);
```

***Note***

*This function should only be called by GUIX.*

**Return values**

| | |
|---|---|
| GX_SUCCESS | Device driver setup is successfully done. |
| GX_FAILURE | Device driver setup failed. |

### 5.2.8.2 Capture Engine Unit (r_ceu)

Modules » Graphics

## Functions

| | |
|---|---|
| fsp_err_t | R_CEU_Open (capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg) |
| fsp_err_t | R_CEU_Close (capture_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CEU_CaptureStart (capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer) |
| fsp_err_t | R_CEU_CallbackSet (capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_CEU_StatusGet (capture_ctrl_t *const p_ctrl, capture_status_t *p_status) |

### Detailed Description

Driver for the CEU peripheral on RA MCUs. This module implements the CAPTURE Interface.

# Overview

The CEU peripheral supports interfacing with external cameras by accepting timing and data signals in order to capture incoming data. A callback is invoked for each V-Sync event, frame of data accepted, or when certain errors occur.

### Features

- Supports 8 or 16-bit camera bus
- Capture images up to 5 MP (2560x1920)
- Capture incoming data directly into a user defined memory location without the need for DTC/DMAC
- Capture specified size 'raw' image data using Data Synchronous Fetch mode (e.g. RGB565, YUV422, etc.)
- Capture binary image data using Data Enable fetch mode (e.g. JPEG)
- Perform basic timing signal validation

# Configuration

### Build Time Configurations for r_ceu

The following build time configurations are defined in fsp_cfg/r_ceu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Graphics > Capture Engine Unit (r_ceu)

This module can be added to the Stacks tab via New Stack > Graphics > Capture Engine Unit

(r_ceu). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_ceu0 | Module name. |
| Input | | | |
| Input > Data Bus Specifications | | | |
| Data Bus Size | MCU Specific Options | | Data bus-width of CEU physical connection. |
| HSYNC Polarity | • High<br>• Low | High | Specify the active polarity of the HSYNC signal. |
| VSYNC Polarity | • High<br>• Low | High | Specify the active polarity of the VSYNC signal. |
| Input > Capture Specifications | | | |
| Input > Capture Specifications > Sample Points | | | |
| Data Sample Point | • Rising edge of the camera clock<br>• Falling edge of the camera clock | Rising edge of the camera clock | Specify the external camera clock transition state for fetching the image data (D15 to D0) from the external module. |
| H-Sync Sample Point | • Rising edge of the camera clock<br>• Falling edge of the camera clock | Rising edge of the camera clock | Specify the external camera clock transition state for capturing H-Sync from the external module. |
| V-Sync Sample Point | • Rising edge of the camera clock<br>• Falling edge of the camera clock | Rising edge of the camera clock | Specify the external camera clock transition state for capturing V-Sync from the external module. |
| Horizontal capture resolution | Value must be an integer. | 640 | Specify the number of horizontal pixels to capture. |
| Vertical capture resolution | Value must be an integer. | 480 | Specify the number of vertical pixels to capture. |
| Horizontal pixel offset | Value must be an integer. | 0 | Number of pixels from H-sync signal up to the |

| | | | |
|---|---|---|---|
| | | | start of a valid data period. |
| Vertical pixel offset | Value must be an integer. | 0 | Specify the vertical line to start capturing image data. |
| Capture Mode | • Data Synchronous Fetch<br>• Data Enable Fetch | Data Synchronous Fetch | Capture mode of incoming data. |
| Output | | | |
| Output > Buffer | | | |
| Data Enable Buffer Size | Value must be a positive integer. | 0 | Specify size of image region available for use by CEU data bus. (Only applicable to Data Enable Fetch capture mode) |
| Byte Swapping | • Swap 8-bit units<br>• Swap 16-bit units<br>• Swap 32-bit units | | Byte swapping configuration. Bytes may be swapped in 8-bit, 16-bit, or 32-bit units. |
| Burst Transfer Mode | • Transfer in 32-byte units<br>• Transfer in 64-byte units<br>• Transfer in 128-byte units<br>• Transfer in 256-byte units | Transfer in 256-byte units | Specify the unit for transferring data to the bus bridge module. |
| Interrupts | | | |
| Interrupts > Selectable CEU Events | | | |
| One-Frame Capture End Event | • Enabled<br>• Disabled | Enabled | Capturing one frame from an external module has finished. |
| Horizontal Sync Event | • Enabled<br>• Disabled | Disabled | Horizontal sync signal was input from an external module. |
| Vertical Sync Event | • Enabled<br>• Disabled | Enabled | Vertical sync signal was input from an external module. |
| CRAM Buffer Overflow Error | • Enabled<br>• Disabled | Enabled | Data overflowed in the CRAM write buffer. |
| H-Sync Validation Error | • Enabled<br>• Disabled | Disabled | The number of configured H-sync |

| | | | |
|---|---|---|---|
| | | | cycles is different than the H-sync cycles input from the external module. |
| V-Sync Validation Error | • Enabled<br>• Disabled | Disabled | The number of configured V-sync cycles is different than the V-sync cycles input from the external module. |
| V-Sync Error | • Enabled<br>• Disabled | Enabled | V-sync has been input while CEU holds data (insufficient vertical-sync front porch). |
| No H-Sync error | • Enabled<br>• Disabled | Enabled | No H-sync signal was input. |
| No V-Sync error | • Enabled<br>• Disabled | Enabled | No V-sync signal was input. |
| Callback | Name must be a valid C symbol. | g_ceu0_user_callback | A user callback function must be provided. This callback is invoked for every successful frame capture as well as other status or error conditions. |
| CEU Interrupt Priority | MCU Specific Options | | Select the CEU interrupt priority. |

### Clock Configuration

The CEU peripheral is clocked both from PCLKA and externally, from the camera module (VIO_CLK). The external input clock (VIO_CLK) should have a frequency at most the same as the CEU operating clock (PCLKA) frequency, with jitter on both sides included. The PCLKA frequency may be set using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

*Note*

*At least 10 external clock cycles (VIO_CLK) must elapse after opening the CEU module, before starting a capture.*

### Pin Configuration

The VIO_CLK pin is a clock input to the MCU and should be connected to the clock output from the camera. The VIO_HD and VIO_VD pins must be connected to the horizontal and vertical sync signal output of the camera respectively. The VIO_D0 to VIO_D15 pins are the data bus input pins and should be connected to the relevant output pins of the camera. For 8-bit camera data bus VIO_D0 to VIO_D7 should be used.

*Note*

*Camera control and serial communication pins must be configured separately and are not controlled by this module.*

### Capture Resolution

- For Data Synchronous Fetch mode, capture size is calculated using the configured Horizontal Resolution, Vertical Resolution, and Bytes Per Pixel.
- For Data Enable Fetch mode, the capture size is controlled by the external module and is defined as a period from a VD rising edge to the VD falling edge. For this mode Horizontal Resolution and Vertical Resolution are not used. However, the maximum capture size is configured using the 'Data Enable Buffer Size' option, which is used to configure the maximum write 'Firewall Address' location.

*Note*

*For Data Enable Fetch mode, the external module must transmit data in 4-byte units.*

### Capture Offset

The blanking period from a horizontal or vertical sync signal differs among external modules. Therefore, the capture start location must be specified in terms of external cycles from the sync signal so that an image can be captured from the valid image area. Some external modules output a horizontal sync signal as a data enable signal. In this case, there is no blanking period so the configured offsets must be cleared to 0.

- The horizontal capture start location must be specified in terms of the number of pixels from a horizontal sync signal.
- The vertical capture start location must be specified in terms of the number of H-cycles from a vertical sync signal.

*Note*

*Capture offset is not used when Data Enable Fetch mode is configured.*

## Usage Notes

### Interrupt Configuration

- CEU V-sync, capture-end, and error interrupts are used by this module for reporting capture status and error events such as overrun, vertical line number setting and other capture errors.

*Note*

*If both a capture complete event and capture error event occur simultaneously, the capture complete event should be disregarded.*

### CEU Setup With External Camera

- Ensure that the memory pointed to by p_buffer is both valid and large enough to store a complete image.
- When Data Synchronous Fetch mode is configured, the amount of space required (in bytes) may be calculated as size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel.
- Ensure that the capture buffer address is 4-byte aligned and the buffer size is divisible by 32 bytes.

*Note*

*Any required configuration for external cameras must be performed by the application.*

# Examples

### Basic Example

This is a basic example of minimal use of the CEU in an application. This example shows how this driver can be used for capturing data from an external I/O device such as an image sensor.

```
bool g_ceu_capture_error;
bool g_ceu_capture_complete;
void ceu_minimal_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ceu_capture_error    = false;
    g_ceu_capture_complete = false;
    err = R_CEU_Open(&g_ceu0_ctrl, &g_ceu0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_CEU_CaptureStart(&g_ceu0_ctrl, g_user_buffer);
    assert(FSP_SUCCESS == err);
 while (!g_ceu_capture_complete && !g_ceu_capture_error)
    {
 /* Wait for capture to complete. */
    }
/* Process image here if capture was successful. */
    err = R_CEU_Close(&g_ceu0_ctrl);
    assert(FSP_SUCCESS == err);
}
void ceu_callback (capture_callback_args_t * p_args)
{
 /* Multiple event flags may be set simultaneously */
 if (p_args->event & (uint32_t) ~(CEU_EVENT_HD | CEU_EVENT_VD | CEU_EVENT_FRAME_END))
    {
 /* Error processing should occur first. Application should not process complete
event if error occurred. */
       g_ceu_capture_error = true;
    }
 else
```

```
    {
 if (p_args->event & CEU_EVENT_VD)
      {
 /* Capture has started. Process V-Sync event. */
      }
 if (p_args->event & CEU_EVENT_FRAME_END)
      {
 /* Capture is complete and no error has occurred */
            g_ceu_capture_complete = true;
      }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | ceu_byte_swapping_t |
| struct | ceu_edge_info_t |
| struct | ceu_extended_cfg_t |
| struct | ceu_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ceu_capture_mode_t |
| enum | ceu_data_bus_size_t |
| enum | ceu_hsync_polarity_t |
| enum | ceu_vsync_polarity_t |
| enum | ceu_burst_transfer_mode_t |
| enum | ceu_event_t |
| enum | ceu_capture_format_t |

## Data Structure Documentation

### ◆ ceu_byte_swapping_t

| struct ceu_byte_swapping_t |
|---|
| |

| Swap bits configuration | | |
|---|---|---|
| Data Fields | | |
| uint8_t | swap_8bit_units: 1 | Byte swapping in 8-bit units. |
| uint8_t | swap_16bit_units: 1 | Byte swapping in 16-bit units. |
| uint8_t | swap_32bit_units: 1 | Byte swapping in 32-bit units. |

### ◆ ceu_edge_info_t

| struct ceu_edge_info_t | | |
|---|---|---|
| Edge information for latching signals | | |
| Data Fields | | |
| uint8_t | dsel: 1 | Sets the edge for fetching the image data (D15 to D0) from an external module. |
| uint8_t | hdsel: 1 | Sets the edge for capturing hd from external module. |
| uint8_t | vdsel: 1 | Sets the edge for capturing vd from external module. |

### ◆ ceu_extended_cfg_t

| struct ceu_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for CEU. | | |
| Data Fields | | |
| ceu_capture_format_t | capture_format | Capture format for incoming data. |
| ceu_data_bus_size_t | data_bus_width | Size of camera data bus. |
| ceu_edge_info_t | edge_info | |
| ceu_hsync_polarity_t | hsync_polarity | Polarity of HSYNC input. |
| ceu_vsync_polarity_t | vsync_polarity | Polarity of VSYNC input. |
| uint32_t | image_area_size | Image capture size. Used when setting firewall address for Data Enable Fetch mode. |
| ceu_byte_swapping_t | byte_swapping | Controls byte swapping in 8-bit, 16-bit and 32-bit units. |
| ceu_burst_transfer_mode_t | burst_mode | Bus transfer data size. |
| uint32_t | interrupts_enabled | Enabled interrupt events bit mask. |
| uint8_t | ceu_ipl | PDC interrupt priority. |
| IRQn_Type | ceu_irq | PDC IRQ number. |

### ◆ ceu_instance_ctrl_t

| struct ceu_instance_ctrl_t |
| --- |
| CEU instance control block. DO NOT INITIALIZE. |

## Enumeration Type Documentation

### ◆ ceu_capture_mode_t

| enum ceu_capture_mode_t | |
| --- | --- |
| Capture mode | |
| Enumerator | |
| CEU_CAPTURE_MODE_SINGLE | Single image capture. |
| CEU_CAPTURE_MODE_CONTINUOUS | Continuous image capture. |

### ◆ ceu_data_bus_size_t

| enum ceu_data_bus_size_t | |
| --- | --- |
| Data bus width | |
| Enumerator | |
| CEU_DATA_BUS_SIZE_8_BIT | Data bus is 8-bit. |
| CEU_DATA_BUS_SIZE_16_BIT | Data bus is 16-bit. |

### ◆ ceu_hsync_polarity_t

| enum ceu_hsync_polarity_t | |
| --- | --- |
| Polarity of input HSYNC signal | |
| Enumerator | |
| CEU_HSYNC_POLARITY_HIGH | HSYNC signal is active high. |
| CEU_HSYNC_POLARITY_LOW | HSYNC signal is active low. |

### ◆ ceu_vsync_polarity_t

| enum ceu_vsync_polarity_t | |
|---|---|
| Polarity of input VSYNC signal | |
| Enumerator | |
| CEU_VSYNC_POLARITY_HIGH | VSYNC signal is active high. |
| CEU_VSYNC_POLARITY_LOW | VSYNC signal is active low. |

### ◆ ceu_burst_transfer_mode_t

| enum ceu_burst_transfer_mode_t | |
|---|---|
| Enumerator | |
| CEU_BURST_TRANSFER_MODE_X1 | Transferred to the bus in 32-byte units */. |
| CEU_BURST_TRANSFER_MODE_X2 | Transferred to the bus in 64-byte units */. |
| CEU_BURST_TRANSFER_MODE_X4 | Transferred to the bus in 128-byte units */. |
| CEU_BURST_TRANSFER_MODE_X8 | Transferred to the bus in 256-byte units */. |

### ◆ ceu_event_t

| enum ceu_event_t | |
|---|---|
| Enumerator | |
| CEU_EVENT_FRAME_END | Frame end event (CPE) |
| CEU_EVENT_HD | (Not Used) HD received (HD) |
| CEU_EVENT_VD | VD received (VD) |
| CEU_EVENT_CRAM_OVERFLOW | Data overflowed in the CRAM buffer (CDTOF) |
| CEU_EVENT_HD_MISMATCH | HD mismatch (IGHS) |
| CEU_EVENT_VD_MISMATCH | VD mismatch (IGVS) |
| CEU_EVENT_VD_ERROR | Invalid VD condition (VBP) |
| CEU_EVENT_FIREWALL | Data write address exceeds firewall (FWF) |
| CEU_EVENT_HD_MISSING | HD was expected but not input (NHD) |
| CEU_EVENT_VD_MISSING | VD was expected but not input (NVD) |

### ◆ ceu_capture_format_t

| enum ceu_capture_format_t | |
|---|---|
| Capture mode for CEU. | |
| Enumerator | |
| CEU_CAPTURE_FORMAT_DATA_SYNCHRONOUS | Raw formatted data. |
| CEU_CAPTURE_FORMAT_DATA_ENABLE | JPG formatted data. |

## Function Documentation

### ◆ R_CEU_Open()

fsp_err_t R_CEU_Open ( capture_ctrl_t *const *p_ctrl*, capture_cfg_t const *const *p_cfg* )

CEU module initialization.

Implements capture_api_t::open

The function provides initial configuration for the CEU module. Further initialization may be performed in capture_api_t::captureStart. This function should be called once prior to calling any other CEU API functions. After the CEU is opened the Open function should not be called again without first calling the Close function.

Example:

```
err = R_CEU_Open(&g_ceu0_ctrl, &g_ceu0_cfg);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more parameters is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_CEU_Close()

fsp_err_t R_CEU_Close ( capture_ctrl_t *const *p_ctrl*)

Stops and closes the transfer interface.

Implements capture_api_t::close

Stops any active captures, clears internal driver state-data, disables interrupts, and powers off the CEU peripheral.

Example:

```
err = R_CEU_Close(&g_ceu0_ctrl);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more parameters is NULL. |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |

### ◆ R_CEU_CaptureStart()

fsp_err_t R_CEU_CaptureStart ( capture_ctrl_t *const *p_ctrl*, uint8_t *const *p_buffer* )

Starts a capture.

Implements capture_api_t::captureStart.

Sets up the interface to transfer data from the CEU into the specifiec buffer. Configures the CEU settings as previously set by the capture_api_t::open API. When a capture is complete the callback registered during capture_api_t::open API call or by capture_api_t::callbackSet API will be called.

Example:

```
err = R_CEU_CaptureStart(&g_ceu0_ctrl, g_user_buffer);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more parameters is NULL. |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |
| FSP_ERR_INVALID_ADDRESS | Invalid buffer address alignment. |
| FSP_ERR_IN_USE | Capture is in progress. |
| FSP_ERR_INVALID_STATE | Capture is pending. |
| FSP_ERR_NOT_INITIALIZED | Callback function has not been set. |

### ◆ R_CEU_CallbackSet()

fsp_err_t R_CEU_CallbackSet ( capture_ctrl_t *const *p_ctrl*, void(*)(capture_callback_args_t *) *p_callback*, void *const *p_context*, capture_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure.

Implements capture_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more parameters is NULL. |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_CEU_StatusGet()

| fsp_err_t R_CEU_StatusGet ( capture_ctrl_t *const *p_ctrl*, capture_status_t * *p_status* ) |
|---|
| Provides the ceu operating status.<br>Implements capture_api_t::statusGet.<br><br>**Return values**<br>FSP_SUCCESS — Initialization was successful.<br>FSP_ERR_ASSERTION — One or more parameters is NULL.<br>FSP_ERR_NOT_OPEN — Open has not been successfully called. |

### 5.2.8.3 D/AVE 2D Port Interface (r_drw)

Modules » Graphics

Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.

# Overview

*Note*

***The D/AVE 2D Port Interface (D1 layer) is a HAL layer for the D/AVE D2 layer API and does not provide any interfaces to the user. Consult the TES Dave2D Driver Documentation for further information on using the D2 API.***

*For cross-platform compatibility purposes the D1 and D2 APIs are not bound by FSP coding guidelines for function names and general module functionality.*

# Configuration

### Build Time Configurations for r_drw

The following build time configurations are defined in fsp_cfg/r_drw_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Allow Indirect Mode | • Enabled<br>• Disabled | Enabled | Enable indirect mode to allow no-copy mode for d2_adddlist (see the TES Dave2D Driver Documentation for details). |
| Memory Allocation | • Default | Default | Set Memory Allocation |

| | | |
|---|---|---|
| | • Custom | to Default to use built-in dynamic memory allocation for the D2 heap. This will use an RTOS heap if configured; otherwise, standard C malloc and free will be used.<br>Set to Custom to define your own allocation scheme for the D2 heap. In this case, the developer will need to define the following functions:<br><br>void * d1_malloc(size_t size)<br>void d1_free(void * ptr) |

**Configurations for Graphics > D/AVE 2D Port Interface (r_drw)**

This module can be added to the Stacks tab via New Stack > Graphics > D/AVE 2D Port Interface (r_drw).

| Configuration | Options | Default | Description |
|---|---|---|---|
| D2 Device Handle Name | Name must be a valid C symbol | d2_handle0 | Set the name for the d2_device handle used when calling D2 layer functions. |
| DRW Interrupt Priority | MCU Specific Options | | Select the DRW_INT (display list completion) interrupt priority. |

**Heap Size**

The D1 port layer allows the D2 driver to allocate memory as needed. There are three ways the driver can accomplish this:

1. Allocate memory using the main heap
2. Allocate memory using a heap provided by an RTOS
3. Allocate memory via user-provided functions

When the "Memory Allocation" configuration option is set to "Default" the driver will use an RTOS implementation if available and the main heap otherwise. Setting the option to "Custom" allows the user to define their own scheme using the following prototypes:

```
void * d1_malloc(size_t size);

void d1_free(void * ptr);
```

Warning

If there is no RTOS-based allocation scheme the main heap will be used. Be sure that it is enabled by setting the "Heap size (bytes)" property under RA Common on the **BSP** tab of the RA Configuration editor.

*Note*

*It is recommended to add 32KB of additional heap space for the D2 driver until the actual usage can be determined in your application.*

### Interrupt

The D1 port includes one interrupt to handle various events like display list completion or bus error. This interrupt is managed internally by the D2 driver and no callback function is available.

## Usage Notes

### Limitations

Developers should be aware of the following limitations when using the DRW engine:

- The DRW module supports two additional interrupt types - bus error and render complete. These interrupts are not needed for D2 layer operation and thus are not supported.
- If the DRW module is stopped during rendering the render will continue once the module is started again. If this behavior is undesirable in your application it is recommended to call d2_flushframe before stopping the peripheral.
- The resolution and size limits for D/AVE 2D are as follows:

| Item | Limit (pixels) |
|---|---|
| Coordinate range | -2048 to 2047 (1:11:4 fixedpoint) |
| Object/primitive size | 1024 x 1024 (10:4 fixedpoint) |
| Framebuffer size | 2048 x 2048 |
| Texture size | 1024 x 1024 |
| Blit size | 2047 x 1024 |

### 5.2.8.4 Display Serial Interface (r_vin)

Modules » Graphics

#### Functions

| | |
|---|---|
| fsp_err_t | R_VIN_Open (capture_ctrl_t *const p_api_ctrl, capture_cfg_t const *const p_cfg) |
| fsp_err_t | R_VIN_Close (capture_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_VIN_CaptureStart (capture_ctrl_t *const p_api_ctrl, uint8_t *const p_buffer) |

| | |
|---|---|
| fsp_err_t | R_VIN_StatusGet (capture_ctrl_t *const p_api_ctrl, capture_status_t *p_status) |

### Detailed Description

Driver for the VIN peripheral on RA MCUs. This module implements the CAPTURE Interface.

# Overview

The VIN peripheral relies on the use of a Camera Serial Interface (CSI-2) Host, physical layer (D-PHY), and supporting sub-systems. Together, these form a high-speed graphics serial bus that formats and writes image data to memory according to the MIPI_CSI and R_VIN configurations.

### Features

The module has one video channel that can control the capture of data into a capture area of up to 4096 × 4096 pixels when scaling is off (2048 × 2048 pixels when scaling is on). It can also provide vertical and horizontal scaling of the data by up to three and two times, respectively.

For captured video data, the VIN provides a color space conversion function from YCbCr-422 to RGB or RGB to YCbCr-422 and a format conversion function from RGB to ARGB.

As the VIN internally generates a field signal, it can capture progressive data.

Supported Data Types:

| Image Format | Input Camera Format (MIPI CSI) | Output Memory Format |
|---|---|---|
| YCbCr-422 8-bit | Supported | Supported |
| YCbCr-422 10-bit | Supported | Supported |
| RGB-565 | | Supported |
| ARGB-1555 | | Supported |
| RGB-888 | Supported | |
| ARGB-8888 | | Supported |
| RAW8 | Supported | Supported |

# Configuration

### Build Time Configurations for r_vin

The following build time configurations are defined in fsp_cfg/r_vin_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Clock Configuration

AXI bus clock (aclk) runs on ICLK and is the main clock for VIN, processing one pixel per clock cycle. In the case of CSI-2, VIN processes the following depths with aclk, so select a clock that is higher than the transfer rate.

- RGB 888: depth = 24 bit
- YCbCr8bit: depth = 16 bit
- YCbCr10bit: depth = 20 bit
- RAW8: depth = 16 bit Example: When setting up for a 720 Mbps 2 lane MIPI CSI transfer rate, with YCbCr 8bit scaling bypass, aclk must be set to a frequency of 720*2/16 = 90 MHz or higher.

### Pin Configuration

VIN does not require pin configuration.

## Usage Notes

### Color Space Conversion

VIN may (optionally) be configured to convert between RGB and YCbCr data.

### RGB to YCbCr Conversion

Formulas are below for the conversion of RGB to YCbCr formats Y = ((C_RtoY × R + C_GtoY × G + C_BtoY × B) × 2 ^ YCLSFT) + Y_Addition Cb = ((C_RtoCb × R + C_GtoCb × G + C_BtoCb × B) × 2 ^ CBCLSFT) + Cb_Addition Cr = ((C_RtoCr × R + C_GtoCr × G + C_BtoCr × B) × 2 ^ CRCLSFT) + Cr_Addition

*Note : YCLSFT, CBCLSFT, and CRCLSFT values are all fixed to 10.*

### YCbCr to RGB Conversion

When enabled, YCbCr to RGB conversion is carried out according to the configured matrix coefficients. The color conversion input is extended to 12 bits by the data extension selection. See User Manual for additional information.

## Examples

### Basic Example

This is a basic example showing the minimum code required to initialize and start the VIN module.

```
void vin_minimal_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = R_VIN_Open(&g_vin0_ctrl, &g_vin0_cfg);
    assert(FSP_SUCCESS == err);
```

```
    USER_CODE_camera_open();          // Prepare camera for capture
    err = R_VIN_CaptureStart(&g_vin0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
    USER_CODE_camera_stream_on();      // Camera begin outputting image data
 capture_status_t capture_status;
    err = R_VIN_StatusGet(&g_vin0_ctrl, &capture_status);
    assert(FSP_SUCCESS == err);
    err = R_VIN_Close(&g_vin0_ctrl);
    assert(FSP_SUCCESS == err);
}
void vin0_callback (capture_callback_args_t * p_args)
{
 vin_event_t event            = (vin_event_t) p_args->event;
 vin_module_status_t    module_status    = (vin_module_status_t)
p_args->event_status;
 vin_interrupt_status_t interrupt_status = (vin_interrupt_status_t)
p_args->interrupt_status;
    switch (event)
    {
 case VIN_EVENT_NOTIFY:
     {
           USER_CODE_vin_process_notify(module_status, interrupt_status);
 if (interrupt_status.bits.frame_complete)
     {
 /* Capture Complete - Process data buffer pointer and index */
               USER_CODE_vin_process_completed_frame(p_args->p_buffer,
module_status.bits.latest_buffer);
           }
 break;
      }
 case VIN_EVENT_ERROR:
     {
 // Process Error based on interrupt and module status;
           USER_CODE_vin_process_error(module_status, interrupt_status);
```

```
 break;
       }
 default:
      {
 /* Do nothing */
 break;
       }
    }
}
void USER_CODE_camera_open (void)
{
 // Pre-transmission Camera Configuration
}
void USER_CODE_camera_stream_on (void)
{
 // Begin Streaming Image Data
}
void USER_CODE_vin_process_error (vin_module_status_t module_status,
vin_interrupt_status_t interrupt_status)
{
 FSP_PARAMETER_NOT_USED(module_status);
 FSP_PARAMETER_NOT_USED(interrupt_status);
 // Process Error
}
void USER_CODE_vin_process_notify (vin_module_status_t module_status,
vin_interrupt_status_t interrupt_status)
{
 FSP_PARAMETER_NOT_USED(module_status);
 FSP_PARAMETER_NOT_USED(interrupt_status);
 // Process Status Notification
}
void USER_CODE_vin_process_completed_frame (uint8_t * p_complete_frame, uint8_t
frame_idx)
{
```

```
 FSP_PARAMETER_NOT_USED(p_complete_frame);
 FSP_PARAMETER_NOT_USED(frame_idx);
 // Process Frame Data
}
```

### Data Structures

| | |
|---|---|
| struct | vin_input_ctrl_t |
| struct | vin_output_ctrl_t |
| struct | vin_conversion_ctrl_t |
| struct | vin_conversion_data_t |
| union | vin_module_status_t |
| union | vin_interrupt_status_t |
| struct | vin_irq_cfg_t |
| struct | vin_interrupt_cfg_t |
| struct | vin_instance_ctrl_t |
| struct | vin_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | vin_interlace_mode_t |
| enum | vin_dithering_mode_t |
| enum | vin_input_format_t |
| enum | vin_data_type_t |
| enum | vin_yuv444_conversion_mode_t |
| enum | vin_pixel_data_clipping_t |
| enum | vin_conversion_mode_t |
| enum | vin_yc_transform_mode_t |
| enum | vin_interpolation_mode_t |
| enum | vin_event_t |

## Data Structure Documentation

### ◆ vin_input_ctrl_t

| struct vin_input_ctrl_t | | |
|---|---|---|
| VIN Input Control Structure | | |
| Data Fields | | |
| union vin_input_ctrl_t | __unnamed__ | |
| struct vin_input_ctrl_t | preclip | |
| union vin_input_ctrl_t | __unnamed__ | |
| union vin_input_ctrl_t | __unnamed__ | |
| uint16_t | image_stride | Image Stride (Setting Unit: pixel) |

### ◆ vin_output_ctrl_t

| struct vin_output_ctrl_t | | |
|---|---|---|
| VIN Output Control Structure | | |
| Data Fields | | |
| uint8_t * | image_buffer[3] | Pointer to Image Buffer Memory Base (64-byte aligned) |

### ◆ vin_conversion_ctrl_t

| struct vin_conversion_ctrl_t |
|---|
| VIN Conversion Control Structure |

### ◆ vin_conversion_data_t

| struct vin_conversion_data_t | | |
|---|---|---|
| VIN Conversion Data Structure | | |
| Data Fields | | |
| uint32_t * | uv_address | Address for YC separation of YCbCr-422 UV data. (64-byte aligned) |
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |

| | | |
|---|---|---|
| union vin_conversion_data_t | __unnamed__ | |
| union vin_conversion_data_t | __unnamed__ | |
| struct vin_conversion_data_t | rgb_to_yuv_conversion_settings [3] | |

### ◆ vin_module_status_t

| union vin_module_status_t |
|---|
| VIN Module Status |

### ◆ vin_interrupt_status_t

| union vin_interrupt_status_t |
|---|
| VIN Interrupt Status |

### ◆ vin_irq_cfg_t

| struct vin_irq_cfg_t | | |
|---|---|---|
| VIN IRQ Configuration | | |
| Data Fields | | |
| uint8_t | ipl | Interrupt priority. |
| IRQn_Type | irq | Interrupt vector number. |

### ◆ vin_interrupt_cfg_t

| struct vin_interrupt_cfg_t | | |
|---|---|---|
| VIN Interrupt Control | | |
| Data Fields | | |
| vin_irq_cfg_t | status | |
| vin_irq_cfg_t | error | |
| union vin_interrupt_cfg_t | __unnamed__ | |
| uint16_t | scanline_compare_value | 12-bit value to be compared with the received line count and trigger interrupt (0: disabled) |

### ◆ vin_instance_ctrl_t

| struct vin_instance_ctrl_t | |
|---|---|
| VIN instance control block. | |
| **Data Fields** | |
| uint32_t | open |
| | Interface is open. |

| | |
|---|---|
| | |
| void(* | p_callback )(capture_callback_args_t *) |
| | Pointer to callback that is called when an adc_event_t occurs. |
| | |
| void * | p_context |
| | Pointer to context to be passed into callback function. |
| | |
| capture_callback_args_t * | p_callback_memory |
| | Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory. |
| | |

### ◆ vin_extended_cfg_t

| struct vin_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for VIN. | | |
| Data Fields | | |
| mipi_csi_instance_t const * | p_mipi_csi_instance | Pointer to mipi physical layer instance. |
| vin_input_ctrl_t | input_ctrl | |
| vin_output_ctrl_t | output_ctrl | |
| vin_conversion_ctrl_t | conversion_ctrl | |
| vin_conversion_data_t | conversion_data | |
| vin_interrupt_cfg_t | interrupt_cfg | |

## Enumeration Type Documentation

### ◆ vin_interlace_mode_t

| enum vin_interlace_mode_t | |
|---|---|
| VIN Interface Mode Type | |
| Enumerator | |
| VIN_INTERLACE_MODE_ODD_FIELD_CAPTURE | Odd-field (field 1) Capture Mode. |
| VIN_INTERLACE_MODE_ODD_EVEN_FIELD_CAPTURE | Odd/Even-field Capture Mode. |
| VIN_INTERLACE_MODE_EVEN_FEILD_CAPTURE | Even-field Cature Mode. |

### ◆ vin_dithering_mode_t

| enum vin_dithering_mode_t | |
|---|---|
| VIN Dithering Mode | |
| Enumerator | |
| VIN_DITHERING_MODE_WITH_ADDITION | Dithereing with Cumulative Addition. |
| VIN_DITHERING_MODE_ORDERED_1 | Ordered Dithering 1. |
| VIN_DITHERING_MODE_ORDERED_2 | Ordered Dithering 2. |

### ◆ vin_input_format_t

| enum vin_input_format_t | |
|---|---|
| VIN Input format | |
| Enumerator | |
| VIN_INPUT_FORMAT_YCBCR422_8_BIT | Input Format 8-Bit YCbCr-422. |
| VIN_INPUT_FORMAT_YCBCR422_10_BIT | Input Format 10-Bit YCbCr-42. |
| VIN_INPUT_FORMAT_RAW8 | Input Format RAW8. |
| VIN_INPUT_FORMAT_RGB888_24_BIT | Input Format 24-Bit RGB888. |

### ◆ vin_data_type_t

| enum vin_data_type_t | |
|---|---|
| VIN Data Type Select | |
| Enumerator | |
| VIN_DATA_TYPE_YUV422_8_BIT | Data Type YUV422 8-bit. |
| VIN_DATA_TYPE_YUV422_10_BIT | Data Type YUV422 10-bit. |
| VIN_DATA_TYPE_RGB888_BIT | Data Type RGB888. |
| VIN_DATA_TYPE_RAW8_BIT | Data Type RAW8. |

### ◆ vin_yuv444_conversion_mode_t

| enum vin_yuv444_conversion_mode_t | |
|---|---|
| VIN Input YUV444 Conversion Mode | |
| Enumerator | |
| VIN_YUV444_CONVERSION_MODE_DATA_EXTEND | The data of CbCr to be interpolated holds the data one pixel before. |
| VIN_YUV444_CONVERSION_MODE_INTERPOLATE | Generates CbCr data to be interpolated. |

### ◆ vin_pixel_data_clipping_t

| enum vin_pixel_data_clipping_t | |
|---|---|
| VIN Pixel Data Clipping | |
| Enumerator | |
| VIN_PIXEL_DATA_CLIPPING_DEFAULT | Default clipping mode. |
| VIN_PIXEL_DATA_CLIPPING_NIBBLE | Lower 4-bits clipping mode. |
| VIN_PIXEL_DATA_CLIPPING_DISABLED | Clipping mode disabled. |

### ◆ vin_conversion_mode_t

| enum vin_conversion_mode_t | |
|---|---|
| VIN Conversion Mode | |
| Enumerator | |
| VIN_CONVERSION_MODE_NONE | Data is not converted before output. |
| VIN_CONVERSION_MODE_RGB_TO_ARGB | RGB input Data is converted to ARGB before output. |
| VIN_CONVERSION_MODE_YC_SEPARATED | Y and CbCr data are separated before output. |

### ◆ vin_yc_transform_mode_t

| enum vin_yc_transform_mode_t | |
|---|---|
| VIN YC Transform Mode | |
| Enumerator | |
| VIN_YC_TRANSFORM_MODE_Y_CBCR | Y and CbCr data are transferred to memory. |
| VIN_YC_TRANSFORM_MODE_Y | Y data is transferred to memory as 8-bit data. |
| VIN_YC_TRANSFORM_MODE_Y10_CBCR | 10-bit Y data and 8-bit CbCr data are transferred to memory |
| VIN_YC_TRANSFORM_MODE_Y10 | 10-bit Y data is transferred to memory |

### ◆ vin_interpolation_mode_t

| enum vin_interpolation_mode_t | |
|---|---|
| VIN Interpolation Mode | |
| Enumerator | |
| VIN_INTERPOLATION_MODE_BILINEAR | Y and CbCr data are transferred to memory. |
| VIN_INTERPOLATION_MODE_NEIGHBOR | Y data is transferred to memory as 8-bit data. |

### ◆ vin_event_t

| enum vin_event_t | |
|---|---|
| VIN instance control block. Implements | |
| Enumerator | |
| VIN_EVENT_NONE | No Event. |
| VIN_EVENT_NOTIFY | Notification Event. |
| VIN_EVENT_ERROR | Error Event. |

## Function Documentation

### ◆ R_VIN_Open()

fsp_err_t R_VIN_Open ( capture_ctrl_t *const *p_api_ctrl*, capture_cfg_t const *const *p_cfg* )

Initialize the VIN peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | One or both of the parameters was NULL. |
| FSP_ERR_ALREADY_OPEN | The instance is already opened. |

### ◆ R_VIN_Close()

fsp_err_t R_VIN_Close ( capture_ctrl_t *const *p_api_ctrl*)

Close VIN and display data instances, disable interrupts, and power-off the module.

- NOTE: Application shall issue stop command to the peripheral and wait at least one full frame before calling CSI_Stop, after ensuring MIPI CSI reception is no longer active by checking data returned by MIPI_CSI GetStatus()

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |

### ◆ R_VIN_CaptureStart()

fsp_err_t R_VIN_CaptureStart ( capture_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_buffer* )

Start video capture.

- NOTE: For best results, camera should not be transmitting data until after VIN capture start has been called.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_IN_USE | The physical interface is currently in use. |
| FSP_ERR_INVALID_STATE | VIN is already in video mode. |
| FSP_ERR_UNSUPPORTED | p_buffer parameter is not supported. |

### ◆ R_VIN_StatusGet()

| fsp_err_t R_VIN_StatusGet ( capture_ctrl_t *const *p_api_ctrl*, capture_status_t * *p_status* ) |
|---|
| Provide information about current VIN status.<br><br>**Return values**<br>FSP_SUCCESS — Information read successfully.<br>FSP_ERR_ASSERTION — p_api_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — Instance is not open. |

## 5.2.8.5 Graphics LCD (r_glcdc)

Modules » Graphics

### Functions

| | |
|---|---|
| fsp_err_t | R_GLCDC_Open (display_ctrl_t *const p_api_ctrl, display_cfg_t const *const p_cfg) |
| fsp_err_t | R_GLCDC_Close (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_Start (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_Stop (display_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_GLCDC_LayerChange (display_ctrl_t const *const p_api_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_BufferChange (display_ctrl_t const *const p_api_ctrl, uint8_t *const framebuffer, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_ColorCorrection (display_ctrl_t const *const p_api_ctrl, display_correction_t const *const p_correction) |
| fsp_err_t | R_GLCDC_ClutUpdate (display_ctrl_t const *const p_api_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer) |
| fsp_err_t | R_GLCDC_ClutEdit (display_ctrl_t const *const p_api_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color) |
| fsp_err_t | R_GLCDC_ColorKeySet (display_ctrl_t const *const p_api_ctrl, display_colorkeying_layer_t key_cfg, display_frame_layer_t layer) |

| | |
|---|---|
| fsp_err_t | R_GLCDC_StatusGet (display_ctrl_t const *const p_api_ctrl, display_status_t *const p_status) |

### Detailed Description

Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface.

# Overview

The GLCDC is a multi-stage graphics output peripheral designed to automatically generate timing and data signals for LCD panels. As part of its internal pipeline the two internal graphics layers can be repositioned, alpha blended, color corrected, dithered and converted to and from a wide variety of pixel formats.

### Features

The following features are available:

| Feature | Options |
|---|---|
| Input color formats | ARGB8888, ARGB4444, ARGB1555, RGB888 (32-bit), RGB565, CLUT 8bpp, CLUT 4bpp, CLUT 1bpp |
| Output color formats | RGB888, RGB666, RGB565, Serial RGB888 (8-bit parallel) |
| Correction processes | Alpha blending, positioning, brightness and contrast, gamma correction, dithering |
| Timing signals | Dot clock, Vsync, Hsync, Vertical and horizontal data enable (DE) |
| Maximum resolution | Up to 2038 x 2043 pixels (dependent on sync signal width) |
| Maximum dot clock | 60MHz for serial RGB mode, 54MHz otherwise |
| Internal clock divisors | 1-9, 12, 16, 24, 32 |
| Interrupts | Line detect (Vblank), Layer 1 underflow, Layer 2 underflow |
| Other functions | Byte-order and endianness control, line repeat function |

# Configuration

### Build Time Configurations for r_glcdc

The following build time configurations are defined in fsp_cfg/r_glcdc_cfg.h:

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected, code for parameter checking is included in the build. |
| Color Correction | • On<br>• Off | Off | If selected, code to adjust brightness, contrast and gamma settings is included in the build. When disabled all color correction configuration options are ignored. |

**Configurations for Graphics > Graphics LCD (r_glcdc)**

This module can be added to the Stacks tab via New Stack > Graphics > Graphics LCD (r_glcdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_display0 | Module name. |
| Interrupts | | | |
| Callback Function | Name must be a valid C symbol | NULL | A user callback function can be defined here. |
| Line Detect Interrupt Priority | MCU Specific Options | | Select the line detect (Vsync) interrupt priority. |
| Underflow 1 Interrupt Priority | MCU Specific Options | | Select the underflow interrupt priority for layer 1. |
| Underflow 2 Interrupt Priority | MCU Specific Options | | Select the underflow interrupt priority for layer 2. |
| Input | | | |
| Input > Graphics Layer 1 | | | |
| Input > Graphics Layer 1 > General | | | |
| Enabled | • Yes<br>• No | Yes | Specify Used if the graphics layer 1 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride. |

| | | | |
|---|---|---|---|
| Horizontal size | Value must be an integer from 16 to 2040 | 480 | Specify the number of horizontal pixels. |
| Vertical size | Value must be an integer from 16 to 2043 | 854 | Specify the number of vertical pixels. |
| Horizontal position | Must be a valid non-negative integer with a maximum configurable value of 4091 | 0 | Specify the horizontal offset in pixels of the graphics layer from the background layer. |
| Vertical position | Must be a valid non-negative integer with a maximum configurable value of 4094 | 0 | Specify the vertical offset in pixels of the graphics layer from the background layer. |
| Color format | • ARGB8888 (32-bit)<br>• RGB888 (32-bit)<br>• RGB565 (16-bit)<br>• ARGB1555 (16-bit)<br>• ARGB4444 (16-bit)<br>• CLUT8 (8-bit)<br>• CLUT4 (4-bit)<br>• CLUT1 (1-bit) | RGB565 (16-bit) | Specify the graphics layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output. |
| Line descending mode | • Enabled<br>• Disabled | Disabled | Select Used if the framebuffer starts from the bottom of the line. |
| Input > Graphics Layer 1 > Framebuffer | | | |
| Framebuffer name | This property must be a valid C symbol | fb_background | Specify the name for the framebuffer for Layer 1. |
| Number of framebuffers | Must be a valid non-negative integer | 2 | Number of framebuffers allocated for Graphics Layer 1. |
| Section for framebuffer allocation | Manual Entry | .sdram_noinit | Specify the section in which to allocate the framebuffer. |
| Input > Graphics Layer 1 > Line Repeat | | | |
| Enable | • On<br>• Off | Off | Select On if the display will be repeated from a smaller section of the framebuffer. |
| Repeat count | Must be a valid non-negative integer with a maximum configurable value of 65535 i.e | 0 | Specify the number of times the image is repeated. |

| | | | |
|---|---|---|---|
| | (vertical size) x (lines repeat times) must be equal to the panel vertical size | | |
| Input > Graphics Layer 1 > Fading | | | |
| Mode | • None<br>• Fade-in<br>• Fade-out | None | Select the fade method. |
| Speed | Value must be an integer from 0 to 255 | 0 | Specify the number of frames for the fading transition to complete. |
| Input > Graphics Layer 2 | | | |
| Input > Graphics Layer 2 > General | | | |
| Enabled | • Yes<br>• No | No | Specify Used if the graphics layer 2 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride. |
| Horizontal size | Value must be be an integer from 16 to 2040 | 480 | Specify the number of horizontal pixels. |
| Vertical size | Value must be be an integer from 16 to 2043 | 854 | Specify the number of vertical pixels. |
| Horizontal position | Must be a valid non-negative integer with a maximum configurable value of 4091 | 0 | Specify the horizontal offset in pixels of the graphics layer from the background layer. |
| Vertical position | Must be a valid non-negative integer with a maximum configurable value of 4094 | 0 | Specify the vertical offset in pixels of the graphics layer from the background layer. |
| Color format | • ARGB8888 (32-bit)<br>• RGB888 (32-bit)<br>• RGB565 (16-bit)<br>• ARGB1555 (16-bit)<br>• ARGB4444 (16-bit)<br>• CLUT8 (8-bit)<br>• CLUT4 (4-bit)<br>• CLUT1 (1-bit) | RGB565 (16-bit) | Specify the graphics layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output. |
| Line descending mode | • Enabled | Disabled | Select Used if the |

| | | | |
|---|---|---|---|
| | • Disabled | | framebuffer starts from the bottom of the line. |
| Input > Graphics Layer 2 > Framebuffer | | | |
| Framebuffer name | This property must be a valid C symbol | fb_foreground | Specify the name for the framebuffer for Layer 2. |
| Number of framebuffers | Must be a valid non-negative integer | 2 | Number of framebuffers allocated for Graphics Layer 2. |
| Section for framebuffer allocation | Manual Entry | .sdram_noinit | Specify the section in which to allocate the framebuffer. |
| Input > Graphics Layer 2 > Line Repeat | | | |
| Enable | • On<br>• Off | Off | Select On if the display will be repeated from a smaller section of the framebuffer. |
| Repeat count | Must be a valid non-negative integer with a maximum configurable value of 65535 i.e (vertical size) x (lines repeat times) must be equal to the panel vertical size | 0 | Specify the number of times the image is repeated. |
| Input > Graphics Layer 2 > Fading | | | |
| Mode | • None<br>• Fade-in<br>• Fade-out | None | Select the fade method. |
| Speed | Value must be an integer from 0 to 255 | 0 | Specify the number of frames for the fading transition to complete. |
| Output | | | |
| Output > Timing | | | |
| Horizontal total cycles | Value must be an integer from 24 to 2047 | 559 | Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system |
| Horizontal active video cycles | Value must be an integer from 16 to 2040 | 480 | Specify the number of active video cycles in a horizontal line (including front and back porch). Set to the number of cycles |

| | | | |
|---|---|---|---|
| | | | defined in the data sheet of LCD panel sheet in your system. |
| Horizontal back porch cycles | Value must be an integer from 5 to 2029 | 5 | Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. |
| Horizontal sync signal cycles | Value must be an integer from 0 to 2046 | 2 | Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. |
| Horizontal sync signal polarity | • Low active<br>• High active | Low active | Select the polarity of Hsync signal to match your system. |
| Vertical total lines | Value must be an integer from 20 to 2047 | 894 | Specify number of total lines in a frame (including front and back porch). |
| Vertical active video lines | Value must be an integer from 16 to 2043 | 854 | Specify the number of active video lines in a frame. |
| Vertical back porch lines | Value must be an integer from 3 to 2030 | 20 | Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. |
| Vertical sync signal lines | Value must be an integer from 0 to 2046 | 3 | Specify the Vsync signal assertion lines in a frame. |
| Vertical sync signal polarity | • Low active<br>• High active | Low active | Select the polarity of Vsync signal to match to your system. |
| Data Enable Signal Polarity | • Low active<br>• High active | High active | Select the polarity of Data Enable signal to |

| | | | |
|---|---|---|---|
| | | | match to your system. |
| Sync edge | • Rising edge<br>• Falling edge | Falling edge | Select the polarity of Sync signals to match to your system. |
| Output > Format | | | |
| Color format | • 24bits RGB888<br>• 18bits RGB666<br>• 16bits RGB565<br>• 8bits serial | 24bits RGB888 | Specify the graphics layer output format to match to your LCD panel. |
| Color order | • RGB<br>• BGR | RGB | Select data order for output signal to LCD panel. |
| Endian | • Little endian<br>• Big endian | Little endian | Select data endianness for output signal to LCD panel. |
| Output > Background | | | |
| Alpha | Value must be an integer from 0 to 255 | 255 | Alpha component of the background color. |
| Red | Value must be an integer from 0 to 255 | 0 | Red component of the background color. |
| Green | Value must be an integer from 0 to 255 | 0 | Green component of the background color. |
| Blue | Value must be an integer from 0 to 255 | 0 | Blue component of the background color. |
| CLUT | | | |
| Enabled | • Yes<br>• No | No | Specify Used if selecting CLUT formats for a graphics layer input format. If used, a buffer (CLUT_buffer) will be automatically generated based on the selected pixel width. |
| Size | Must be a valid non-negative integer with a maximum configurable value of 256 | 256 | Specify the number of entries for the CLUT source data buffer. Each entry consumes 4 bytes (1 word). |
| TCON | | | |
| Hsync pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON1 | Select the TCON pin used for the Hsync signal to match to your system. |

| | | | |
|---|---|---|---|
| Vsync pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON0 | Select TCON pin used for Vsync signal to match to your system. |
| Data enable (DE) pin select | • Not used<br>• LCD_TCON0<br>• LCD_TCON1<br>• LCD_TCON2<br>• LCD_TCON3 | LCD_TCON2 | Select TCON pin used for DataEnable signal to match to your system. |
| Panel clock source | • Internal clock (GLCDCLK)<br>• External clock (LCD_EXTCLK) | Internal clock (GLCDCLK) | Choose between an internal GLCDCLK generated from PCLKA or an external clock provided to the LCD_EXTCLK pin. |
| Panel clock division ratio | Refer to the RA Configuration tool for available options. | 1/8 | Select the clock source divider value. |
| Color Correction | | | |
| Color Correction > Brightness | | | |
| Enabled | • Yes<br>• No | No | Enable brightness color correction. |
| Red channel | Value must be an integer from 0 to 1023 | 512 | Red component of the brightness calibration. This value is divided by 512 to determine gain. |
| Green channel | Value must be an integer from 0 to 1023 | 512 | Green component of the brightness calibration. This value is divided by 512 to determine gain. |
| Blue channel | Value must be an integer from 0 to 1023 | 512 | Blue component of the brightness calibration. This value is divided by 512 to determine gain. |
| Color Correction > Contrast | | | |
| Enabled | • Yes<br>• No | No | Enable contrast color correction. |
| Red channel gain | Value must be an integer from 0 to 255 | 128 | Red component of the contrast calibration. This value is divided by 128 to determine gain. |
| Green channel gain | Value must be an integer from 0 to 255 | 128 | Green component of the contrast calibration. This value is divided by 128 to |

| | | | |
|---|---|---|---|
| | | | determine gain. |
| Blue channel gain | Value must be an integer from 0 to 255 | 128 | Blue component of the contrast calibration. This value is divided by 128 to determine gain. |
| Color Correction > Gamma | | | |
| Color Correction > Gamma > Tables | | | |
| Color Correction > Gamma > Tables > Red | | | |
| Color Correction > Gamma > Tables > Red > Gain | | | |
| 0 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 1 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 2 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 3 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 4 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 5 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 6 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 7 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain |

| | | | |
|---|---|---|---|
| | | | of 0 and 1.999, respectively). |
| 8 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 9 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 10 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 11 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 12 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 13 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 14 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 15 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| Color Correction > Gamma > Tables > Red > Threshold | | | |
| 1 | Value must be an integer from 0 to 1023 | 64 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 2 | Value must be an | 128 | Enter a threshold value |

| | | | |
|---|---|---|---|
| | integer from 0 to 1023 | | between the surrounding values less than or equal to 1023. |
| 3 | Value must be an integer from 0 to 1023 | 192 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 4 | Value must be an integer from 0 to 1023 | 256 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 5 | Value must be an integer from 0 to 1023 | 320 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 6 | Value must be an integer from 0 to 1023 | 384 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 7 | Value must be an integer from 0 to 1023 | 448 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 8 | Value must be an integer from 0 to 1023 | 512 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 9 | Value must be an integer from 0 to 1023 | 576 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 10 | Value must be an integer from 0 to 1023 | 640 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 11 | Value must be an integer from 0 to 1023 | 704 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 12 | Value must be an integer from 0 to 1023 | 768 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 13 | Value must be an integer from 0 to 1023 | 832 | Enter a threshold value between the surrounding values less than or equal to 1023. |

| | | | |
|---|---|---|---|
| 14 | Value must be an integer from 0 to 1023 | 896 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 15 | Value must be an integer from 0 to 1023 | 960 | Enter a threshold value between the surrounding values less than or equal to 1023. |

Color Correction > Gamma > Tables > Green

Color Correction > Gamma > Tables > Green > Gain

| | | | |
|---|---|---|---|
| 0 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 1 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 2 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 3 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 4 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 5 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 6 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 7 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain |

| | | | |
|---|---|---|---|
| | | | of 0 and 1.999, respectively). |
| 8 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 9 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 10 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 11 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 12 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 13 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 14 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 15 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| Color Correction > Gamma > Tables > Green > Threshold | | | |
| 1 | Value must be an integer from 0 to 1023 | 64 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 2 | Value must be an | 128 | Enter a threshold value |

| | | | |
|---|---|---|---|
| | integer from 0 to 1023 | | between the surrounding values less than or equal to 1023. |
| 3 | Value must be an integer from 0 to 1023 | 192 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 4 | Value must be an integer from 0 to 1023 | 256 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 5 | Value must be an integer from 0 to 1023 | 320 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 6 | Value must be an integer from 0 to 1023 | 384 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 7 | Value must be an integer from 0 to 1023 | 448 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 8 | Value must be an integer from 0 to 1023 | 512 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 9 | Value must be an integer from 0 to 1023 | 576 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 10 | Value must be an integer from 0 to 1023 | 640 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 11 | Value must be an integer from 0 to 1023 | 704 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 12 | Value must be an integer from 0 to 1023 | 768 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 13 | Value must be an integer from 0 to 1023 | 832 | Enter a threshold value between the surrounding values less than or equal to 1023. |

| | | | |
|---|---|---|---|
| 14 | Value must be an integer from 0 to 1023 | 896 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 15 | Value must be an integer from 0 to 1023 | 960 | Enter a threshold value between the surrounding values less than or equal to 1023. |

Color Correction > Gamma > Tables > Blue

Color Correction > Gamma > Tables > Blue > Gain

| | | | |
|---|---|---|---|
| 0 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 1 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 2 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 3 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 4 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 5 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 6 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 7 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain |

| | | | |
|---|---|---|---|
| | | | of 0 and 1.999, respectively). |
| 8 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 9 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 10 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 11 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 12 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 13 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 14 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| 15 | Value must be an integer from 0 to 2047 | 1024 | Enter a gain value from 0 to 2047 (corresponding to gain of 0 and 1.999, respectively). |
| Color Correction > Gamma > Tables > Blue > Threshold | | | |
| 1 | Value must be an integer from 0 to 1023 | 64 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 2 | Value must be an | 128 | Enter a threshold value |

| | | | |
|---|---|---|---|
| | integer from 0 to 1023 | | between the surrounding values less than or equal to 1023. |
| 3 | Value must be an integer from 0 to 1023 | 192 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 4 | Value must be an integer from 0 to 1023 | 256 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 5 | Value must be an integer from 0 to 1023 | 320 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 6 | Value must be an integer from 0 to 1023 | 384 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 7 | Value must be an integer from 0 to 1023 | 448 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 8 | Value must be an integer from 0 to 1023 | 512 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 9 | Value must be an integer from 0 to 1023 | 576 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 10 | Value must be an integer from 0 to 1023 | 640 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 11 | Value must be an integer from 0 to 1023 | 704 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 12 | Value must be an integer from 0 to 1023 | 768 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 13 | Value must be an integer from 0 to 1023 | 832 | Enter a threshold value between the surrounding values less than or equal to 1023. |

| | | | |
|---|---|---|---|
| 14 | Value must be an integer from 0 to 1023 | 896 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| 15 | Value must be an integer from 0 to 1023 | 960 | Enter a threshold value between the surrounding values less than or equal to 1023. |
| Red | • On<br>• Off | Off | Enable gamma color correction for the red channel. |
| Green | • On<br>• Off | Off | Enable gamma color correction for the green channel. |
| Blue | • On<br>• Off | Off | Enable gamma color correction for the blue channel. |
| Table Mode | • Constant<br>• Variable | Variable | Set to Constant to override the automatically-generated RAM gamma tables with a constant declaration using the provided values. |
| Process order | • Brightness/contrast first<br>• Gamma first | Brightness/contrast first | Select the color correction processing order. |
| Dithering | | | |
| Enabled | • Yes<br>• No | No | Enable dithering to reduce the effect of color banding. |
| Mode | • Truncate<br>• Round off<br>• 2x2 Pattern | Truncate | Select the dithering mode. |
| Pattern A | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Pattern B | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Pattern C | • Pattern 00<br>• Pattern 01<br>• Pattern 10<br>• Pattern 11 | Pattern 11 | Select the dithering pattern. |
| Pattern D | • Pattern 00 | Pattern 11 | Select the dithering |

- Pattern 01
- Pattern 10
- Pattern 11

pattern.

### Clock Configuration

The peripheral clock for this module is PCLKA.

The dot clock is typically generated from the PLL with a maximum output frequency of 54 MHz in most pixel formats (60 MHz for serial RGB). Optionally, a clock signal can be provided to the LCD_EXTCLK pin for finer framerate control (60 MHz maximum input). With either clock source dividers of 1-9, 12, 16, 24 and 32 may be used. Clocks must be initialized and settled prior to starting this module.

### Pin Configuration

This module controls a variety of pins necessary for LCD data and timing signal output:

| Pin Name | Function | Notes |
|---|---|---|
| LCD_EXTCLK | External clock signal input | The maximum input clock frequency is 60MHz. |
| LCD_CLK | Dot clock output | The maximum output frequency is 54MHz (60MHz in serial RGB mode). |
| LCD_DATAn | Pixel data output lines | Pin assignment and color order is based on the output block configuration. See "Output Control for Data Format" in the GLCDC section of the relevant hardware manual for details. |
| LCD_TCONn | Panel timing signal output | These pins can be configured to output vertical and horizontal synchronization and data valid signals. |

*Note*

*There are two banks of pins listed for the GLCDC in the RA6M3 User's Manual (_A and _B). In most cases the _B bank will be used as _A conflicts with SDRAM pins. In either case, it is generally recommended to only use pins from only one bank at a time as this allows for superior signal routing both inside and outside the package. If _A and _B pins must be mixed be sure to note the timing precision penalty detailed in Table 60.33 in in the RA6M3 User's Manual.*

# Usage Notes

### Overview

The GLCDC peripheral is a combination of several sub-peripherals that form a pixel data processing pipeline. Each block passes pixel data to the next but otherwise they are disconnected from one another - in other words, changing timing block parameters does not affect the output generation block configuration and vice versa.

### Initial Configuration

During R_GLCDC_Open all configured parameters are set in the GLCDC peripheral fully preparing it for operation. Once opened, calling R_GLCDC_Start is typically all that is needed for basic operation. Background generation, timing and output parameters are not configurable at runtime, though layer control and color correction options can be altered.

### Framebuffer Allocation

The framebuffer should be allocated in the highest-speed region available without displacing the stack, heap and other program-critical structures. Regardless of the placement two rules must be followed to ensure correct operation of the GLCDC:

- The framebuffer must be aligned on a 64-byte boundary
- The horizontal stride of the buffer must be a multiple of 64 bytes

*Note*

*Framebuffers allocated through the RA Configuration tool automatically follow the alignment and size requirements.*

If your framebuffer will be placed in internal memory it is recommended to avoid any high-speed RAM regions as there is typically no speed advantage for doing so. In particular, it is important to ensure the framebuffer does not push the stack or any heaps outside of high-speed RAM areas to preserve CPU performance.

### Graphics Layers and Timing Parameters

The GLCDC synthesizes graphics data through two configurable graphics layers onto a background layer. The background is used as a solid-color canvas upon which to composite data from the graphics layers. The two graphics layers are blended on top of each other (Layer 2 above Layer 1) and overlaid on the background layer based on their individual configuration. The placement of the layers (as well as LCD timing parameters) are detailed in Figure 1. The colors of the dimensions indicate which element of the display_cfg_t struct is being referenced - for example, the width of the background layer would be [display_cfg].output.htiming.display_cyc as shown in the figure below.

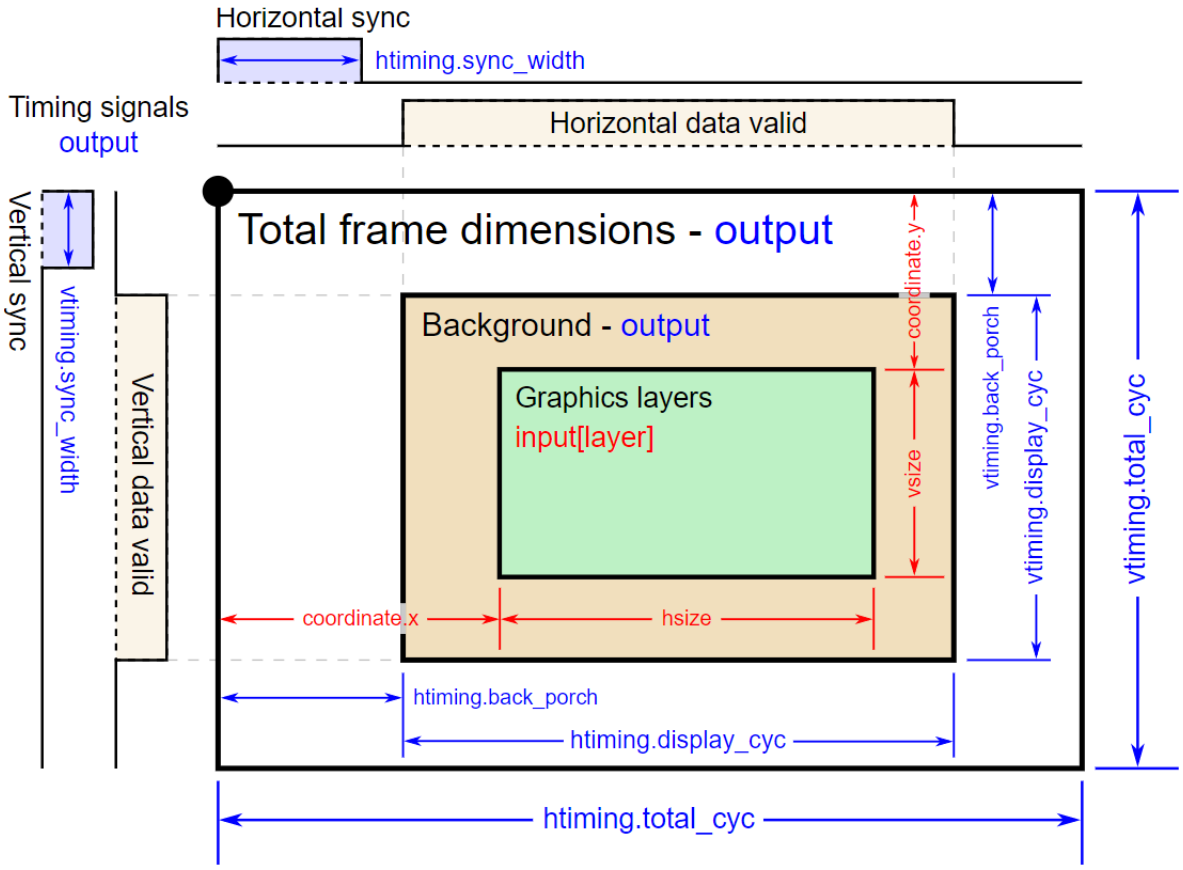



Figure 255: GLCDC layers and timing

*Note*

*The data enable signal (if configured) is the logical AND of the horizontal and vertical data valid signals. In the GLCDC layers and timing figure, only one graphics layer is shown for simplicity. Additionally, in most applications the graphics layer(s) will be the same dimensions as the background layer.*

### Runtime Configuration Options

*Note*

*All runtime configurations detailed below are also automatically configured during R_GLCDC_Open based on the options selected in the RA Configuration editor.*

### Blend processing

Control of layer positioning, alpha blending and fading is possible at runtime via R_GLCDC_LayerChange. This function takes a display_runtime_cfg_t parameter which contains the same input and layer elements as the display_cfg_t control block. Refer to the documentation for display_runtime_cfg_t as well as the Examples below to see what options are configurable.

### Brightness and contrast

Brightness and contrast correction can be controlled through R_GLCDC_ColorCorrection. The display_correction_t parameter is used to control enabling, disabling and gain values for both corrections as shown below:

```
display_correction_t correction;
/* Brightness values are 0-1023 with +512 offset being neutral */
   correction.brightness.r = 512;
   correction.brightness.g = 512;
   correction.brightness.b = 512;
/* Contrast values are 0-255 representing gain of 0-2 (128 is gain of 1) */
   correction.contrast.r = 128;
   correction.contrast.g = 128;
   correction.contrast.b = 128;
/* Brightness and contrast correction can be enabled or disabled independent of one
another */
   correction.brightness.enable = true;
   correction.contrast.enable   = true;
/* Enable correction */
R_GLCDC_ColorCorrection(&g_disp_ctrl, &correction);
```

### Color Look-Up Table (CLUT) Modes

The GLCDC supports 1-, 4- and 8-bit color look-up table (CLUT) formats for input pixel data. By using these modes the framebuffer size in memory can be reduced significantly, allowing even high-resolution displays to be buffered in on-chip SRAM. To enable CLUT modes for a layer the color format must be set to a CLUT mode (either at startup or through R_GLCDC_LayerChange) in addition to filling the CLUT as appropriate via R_GLCDC_ClutUpdate as shown below:

```
/* Basic 4-bit (16-color) CLUT definition */
   uint32_t clut_4[16] =
   {
      0xFF000000,                  // Black
      0xFFFFFFFF,                  // White
      0xFF0000FF,                  // Blue
      0xFF0080FF,                  // Turquoise
      0xFF00FFFF,                  // Cyan
      0xFF00FF80,                  // Mint Green
      0xFF00FF00,                  // Green
      0xFF80FF00,                  // Lime Green
      0xFFFFFF00,                  // Yellow
      0xFFFF8000,                  // Orange
```

```
      0xFFFF0000,                    // Red
      0xFFFF0080,                    // Pink
      0xFFFF00FF,                    // Magenta
      0xFF8000FF,                    // Purple
      0xFF808080,                    // Gray
      0x00000000                      // Transparent
   };
/* Define the CLUT configuration */
display_clut_cfg_t clut_cfg =
   {
      .start = 0,
      .size   = 16,
      .p_base = clut_4
   };
/* Update the CLUT in the GLCDC */
R_GLCDC_ClutUpdate(&g_disp_ctrl, &clut_cfg, DISPLAY_FRAME_LAYER_1);
```

*Note*

*If individual elements of the CLUT must be changed or if elements must be changed one at a time (for instance, when using emWin) it is recommended to use R_GLCDC_ClutEdit to avoid repeated memcpy operations.*

### Other Configuration Options

### Gamma correction

Gamma correction is performed based on a gain curved defined in the RA Configuration editor. Each point on the curve is defined by a threshold and a gain value - each gain value represents a multiplier from 0x-2x (set as 0-2047) that sets the Y-value of the slope of the gain curve, while each threshold interval sets the X-value respectively. For a more detailed explanation refer to Figure "Calculation of gamma correction value" and the related description above it in the GLCDC section of the relevant hardware manual.

When setting threshold values three rules must be followed:

- Each threshold value must be greater than the previous value
- Threshold values must be greater than zero and less than 1024
- Threshold values can equal the previous value only if they are 1023 (maximum)

*Note*

*Gamma correction can only be applied via R_GLCDC_Open.*

### Dithering

Dithering is a method of pixel blending that allows for smoother transitions between colors when

using a limited palette. A full description of dithering is outside the scope of this document. For more information on the pattern settings and how to configure them refer to the Figures "Configuration of dither correction block" and "Addition value selection method for 2x2 pattern dither" in the GLCDC section of the relevant hardware manual.

### Maximum Resolution

Though the GLCDC is capable of outputting resolutions in excess of HD, maximum clockspeeds and bus throughput limit what can be realistically achieved. Below is a table of maximum recommended widescreen resolutions at various input color depths **based on clock and bus limits at 60 FPS.**

| Device | 8bpp | 16bpp | 24/32bpp |
|---|---|---|---|
| RA6M3 | 960x540 (qHD) | 800x480 (WVGA) | 640x360 (nHD) |
| RA8 | 1920x1200 (WUXGA) | 1280x800 (WXGA) | 960x540 (qHD) |

*Note*

*The above values are estimated conservatively based on effective SDRAM bus throughput at a standard framerate. To calculate estimated maximum values for other situations the following equations can be used:*
*bytes per frame = (SDCLK speed (Hz) * SDRAM bus width / 8) / framerate*
*width = sqrt(16/9 * (bytes per frame) / (pixel bit depth / 8))*
*height = 9/16 * width*
*These equations provide values that are theoretically possible but may or may not be attainable depending on a number of other factors. It is the responsibility of the developer to test and confirm performance on each system to determine a suitable panel size for the application.*

Warning

The maximum resolution for GLCDC may exceed the limits for single texture or primitive rending operations with the DRW peripheral (D/AVE 2D). See the Limitations section in the DRW Usage Notes for details.

### Limitations

Developers should be aware of the following limitations when using the GLCDC API:

- Due to a limitation of the GLCDC hardware, if the horizontal back porch is less than the number of pixels in a graphics burst read (64 bytes) for a layer and the layer is positioned at a negative X-value then the layer X-position will be locked to the nearest 64-byte boundary, rounded toward zero.
- The GLCDC peripheral offers a chroma-key function that can be used to perform a green-screen-like color replacement. This functionality is not exposed through the GLCDC API. See the descriptions for GR1_AB7/GR2_AB7 through GR1_AB9/GR2_AB9 in the device's User's Manual for further details.
- Use of R_GLCDC_ClutUpdate and R_GLCDC_ClutEdit may not be mixed on the same frame.

## Examples

### Basic Example

This is a basic example showing the minimum code required to initialize and start the GLCDC module. If the entire display can be drawn within the vertical blanking period no further code may be necessary.

```
void glcdc_init (void)
{
 fsp_err_t err;
 // Open the GLCDC driver
    err = R_GLCDC_Open(&g_disp_ctrl, &g_disp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 // Start display output
    err = R_GLCDC_Start(&g_disp_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Layer Transitions

This example demonstrates how to set up and execute both a sliding and fading layer transition. This is most useful in static image transition scenarios as switching between two actively-drawing graphics layers may require up to four framebuffers to eliminate tearing.

```
volatile uint32_t g_vsync_count = 0;
/* Callback function for GLCDC interrupts */
static void glcdc_callback (display_callback_args_t * p_args)
{
 if (p_args->event == DISPLAY_EVENT_LINE_DETECTION)
    {
       g_vsync_count++;
    }
}
/* Simple wait that returns 1 if no Vsync happened within the timeout period */
uint8_t vsync_wait (void)
{
    uint32_t timeout_timer = GLCDC_VSYNC_TIMEOUT;
    g_vsync_count = 0;
 while (!g_vsync_count && --timeout_timer)
    {
 /* Spin here until DISPLAY_EVENT_LINE_DETECTION callback or timeout */
    }
```

```
 return timeout_timer ? 0 : 1;
}
/* Initiate a fade on Layer 2
 *
 * Parameters:
 * direction True for fade in, false for fade out
 * speed number of frames over which to fade
 */
void glcdc_layer_transition_fade (display_runtime_cfg_t * disp_rt_cfg, bool
direction, uint16_t speed)
{
 fsp_err_t err;
 if (direction)
    {
 /* Set the runtime struct to the desired buffer */
       disp_rt_cfg->input.p_base       = (uint32_t *) g_framebuffer_1;
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEIN;
    }
 else
    {
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEOUT;
    }
 /* Ensure speed is at least 1 frame */
 if (!speed)
    {
       speed = 1;
    }
 /* Set the fade speed to the desired change in alpha per frame */
    disp_rt_cfg->layer.fade_speed = UINT8_MAX / speed;
 /* Initiate the fade (will start on the next Vsync) */
    err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg, DISPLAY_FRAME_LAYER_2);
    assert(FSP_SUCCESS == err);
}
/* Slide Layer 1 out to the left while sliding Layer 2 in from the right */
```

```
void glcdc_layer_transition_sliding (display_runtime_cfg_t * disp_rt_cfg_in,
display_runtime_cfg_t * disp_rt_cfg_out)
{
 fsp_err_t err;
 /* Set the config for the incoming layer to be just out of bounds on the right side
*/
    disp_rt_cfg_in->input.p_base       = (uint32_t *) g_framebuffer_1;
    disp_rt_cfg_in->layer.coordinate.x = DISPLAY_WIDTH;
 /* Move layer 1 out and layer 2 in at a fixed rate of 4 pixels per frame */
 for (int32_t x = disp_rt_cfg_in->layer.coordinate.x; x >= 0; x -= 4)
    {
 /* Wait for a Vsync before starting */
       vsync_wait();
 /* Set the X-coordinate of both layers then update them */
       disp_rt_cfg_out->layer.coordinate.x = (int16_t) (x - DISPLAY_WIDTH);
       disp_rt_cfg_in->layer.coordinate.x = (int16_t) x;
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_out, DISPLAY_FRAME_LAYER_1
);
      assert(FSP_SUCCESS == err);
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_in, DISPLAY_FRAME_LAYER_2
);
      assert(FSP_SUCCESS == err);
    }
}
```

### Double-Buffering

Using a double-buffer allows one to be output to the LCD while the other is being drawn to memory, eliminating tearing and in some cases reducing bus load. The following is a basic example showing integration of the line detect (Vsync) interrupt to set the timing for buffer swapping and drawing.

```
/* User-defined function to draw the current display to a framebuffer */
void display_draw (uint8_t * framebuffer)
{
 FSP_PARAMETER_NOT_USED(framebuffer);
 /* Draw buffer here */
```

```
}
/* This function is an example of a basic double-buffered display thread */
void display_thread (void)
{
    uint8_t * p_framebuffer = NULL;
 fsp_err_t err;
 /* Initialize and start the R_GLCDC module */
    glcdc_init();
 while (1)
    {
 /* Swap the active framebuffer */
       p_framebuffer = (p_framebuffer == g_framebuffer_0) ? g_framebuffer_1 :
g_framebuffer_0;
 /* Draw the new framebuffer now */
       display_draw(p_framebuffer);
 /* Now that the framebuffer is ready, update the GLCDC buffer pointer on the next
Vsync */
       err = R_GLCDC_BufferChange(&g_disp_ctrl, p_framebuffer, DISPLAY_FRAME_LAYER_1
);
     assert(FSP_SUCCESS == err);
 /* Wait for a Vsync event */
       vsync_wait();
    }
}
```

## Data Structures

| | |
|---|---|
| struct | glcdc_instance_ctrl_t |
| struct | glcdc_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | glcdc_clk_src_t |
| enum | glcdc_panel_clk_div_t |
| enum | glcdc_tcon_pin_t |
| enum | glcdc_bus_arbitration_t |

| | |
|---|---|
| enum | glcdc_correction_proc_order_t |
| enum | glcdc_tcon_signal_select_t |
| enum | glcdc_clut_plane_t |
| enum | glcdc_dithering_mode_t |
| enum | glcdc_dithering_pattern_t |
| enum | glcdc_input_interface_format_t |
| enum | glcdc_output_interface_format_t |
| enum | glcdc_dithering_output_format_t |

## Data Structure Documentation

### ◆ glcdc_instance_ctrl_t

| struct glcdc_instance_ctrl_t |
|---|
| Display control block. DO NOT INITIALIZE. |

### ◆ glcdc_extended_cfg_t

| struct glcdc_extended_cfg_t | | |
|---|---|---|
| GLCDC hardware specific configuration | | |
| Data Fields | | |
| glcdc_tcon_pin_t | tcon_hsync | GLCDC TCON output pin select. |
| glcdc_tcon_pin_t | tcon_vsync | GLCDC TCON output pin select. |
| glcdc_tcon_pin_t | tcon_de | GLCDC TCON output pin select. |
| glcdc_correction_proc_order_t | correction_proc_order | Correction control route select. |
| glcdc_clk_src_t | clksrc | Clock Source selection. |
| glcdc_panel_clk_div_t | clock_div_ratio | Clock divide ratio for dot clock. |
| glcdc_dithering_mode_t | dithering_mode | Dithering mode. |
| glcdc_dithering_pattern_t | dithering_pattern_A | Dithering pattern A. |
| glcdc_dithering_pattern_t | dithering_pattern_B | Dithering pattern B. |
| glcdc_dithering_pattern_t | dithering_pattern_C | Dithering pattern C. |
| glcdc_dithering_pattern_t | dithering_pattern_D | Dithering pattern D. |
| void * | phy_layer | Alternate PHY layer, such as MIPI DSI. |

## Enumeration Type Documentation

### ◆ glcdc_clk_src_t

| enum glcdc_clk_src_t | |
|---|---|
| Clock source select | |
| Enumerator | |
| GLCDC_CLK_SRC_INTERNAL | Internal. |
| GLCDC_CLK_SRC_EXTERNAL | External. |

### ◆ glcdc_panel_clk_div_t

| enum glcdc_panel_clk_div_t | |
|---|---|
| Clock frequency division ratio | |
| Enumerator | |
| GLCDC_PANEL_CLK_DIVISOR_1 | Division Ratio 1/1. |
| GLCDC_PANEL_CLK_DIVISOR_2 | Division Ratio 1/2. |
| GLCDC_PANEL_CLK_DIVISOR_3 | Division Ratio 1/3. |
| GLCDC_PANEL_CLK_DIVISOR_4 | Division Ratio 1/4. |
| GLCDC_PANEL_CLK_DIVISOR_5 | Division Ratio 1/5. |
| GLCDC_PANEL_CLK_DIVISOR_6 | Division Ratio 1/6. |
| GLCDC_PANEL_CLK_DIVISOR_7 | Division Ratio 1/7. |
| GLCDC_PANEL_CLK_DIVISOR_8 | Division Ratio 1/8. |
| GLCDC_PANEL_CLK_DIVISOR_9 | Division Ratio 1/9. |
| GLCDC_PANEL_CLK_DIVISOR_12 | Division Ratio 1/12. |
| GLCDC_PANEL_CLK_DIVISOR_16 | Division Ratio 1/16. |
| GLCDC_PANEL_CLK_DIVISOR_24 | Division Ratio 1/24. |
| GLCDC_PANEL_CLK_DIVISOR_32 | Division Ratio 1/32. |

### ◆ glcdc_tcon_pin_t

| enum glcdc_tcon_pin_t | |
|---|---|
| LCD TCON output pin select | |
| Enumerator | |
| GLCDC_TCON_PIN_NONE | No output. |
| GLCDC_TCON_PIN_0 | LCD_TCON0. |
| GLCDC_TCON_PIN_1 | LCD_TCON1. |
| GLCDC_TCON_PIN_2 | LCD_TCON2. |
| GLCDC_TCON_PIN_3 | LCD_TCON3. |

### ◆ glcdc_bus_arbitration_t

| enum glcdc_bus_arbitration_t | |
|---|---|
| Bus Arbitration setting | |
| Enumerator | |
| GLCDC_BUS_ARBITRATION_ROUNDROBIN | Round robin. |
| GLCDC_BUS_ARBITRATION_FIX_PRIORITY | Fixed. |

### ◆ glcdc_correction_proc_order_t

| enum glcdc_correction_proc_order_t | |
|---|---|
| Correction circuit sequence control | |
| Enumerator | |
| GLCDC_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA | Brightness -> contrast -> gamma correction. |
| GLCDC_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST | Gamma correction -> brightness -> contrast. |

### ◆ glcdc_tcon_signal_select_t

| enum glcdc_tcon_signal_select_t | |
|---|---|
| Timing signals for driving the LCD panel | |
| Enumerator | |
| GLCDC_TCON_SIGNAL_SELECT_STVA_VS | STVA/VS. |
| GLCDC_TCON_SIGNAL_SELECT_STVB_VE | STVB/VE. |
| GLCDC_TCON_SIGNAL_SELECT_STHA_HS | STH/SP/HS. |
| GLCDC_TCON_SIGNAL_SELECT_STHB_HE | STB/LP/HE. |
| GLCDC_TCON_SIGNAL_SELECT_DE | DE. |

### ◆ glcdc_clut_plane_t

| enum glcdc_clut_plane_t | |
|---|---|
| Clock phase adjustment for serial RGB output | |
| Enumerator | |
| GLCDC_CLUT_PLANE_0 | GLCDC CLUT plane 0. |
| GLCDC_CLUT_PLANE_1 | GLCDC CLUT plane 1. |

### ◆ glcdc_dithering_mode_t

| enum glcdc_dithering_mode_t | |
|---|---|
| Dithering mode | |
| Enumerator | |
| GLCDC_DITHERING_MODE_TRUNCATE | No dithering (truncate) |
| GLCDC_DITHERING_MODE_ROUND_OFF | Dithering with round off. |
| GLCDC_DITHERING_MODE_2X2PATTERN | Dithering with 2x2 pattern. |

### ◆ glcdc_dithering_pattern_t

| enum glcdc_dithering_pattern_t | |
|---|---|
| Dithering mode | |
| Enumerator | |
| GLCDC_DITHERING_PATTERN_00 | 2x2 pattern '00' |
| GLCDC_DITHERING_PATTERN_01 | 2x2 pattern '01' |
| GLCDC_DITHERING_PATTERN_10 | 2x2 pattern '10' |
| GLCDC_DITHERING_PATTERN_11 | 2x2 pattern '11' |

### ◆ glcdc_input_interface_format_t

| enum glcdc_input_interface_format_t | |
|---|---|
| Output interface format | |
| Enumerator | |
| GLCDC_INPUT_INTERFACE_FORMAT_RGB565 | Input interface format RGB565. |
| GLCDC_INPUT_INTERFACE_FORMAT_RGB888 | Input interface format RGB888. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB1555 | Input interface format ARGB1555. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB4444 | Input interface format ARGB4444. |
| GLCDC_INPUT_INTERFACE_FORMAT_ARGB8888 | Input interface format ARGB8888. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT8 | Input interface format CLUT8. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT4 | Input interface format CLUT4. |
| GLCDC_INPUT_INTERFACE_FORMAT_CLUT1 | Input interface format CLUT1. |

### ◆ glcdc_output_interface_format_t

| enum glcdc_output_interface_format_t | |
|---|---|
| Output interface format | |
| Enumerator | |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB888 | Output interface format RGB888. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB666 | Output interface format RGB666. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_RGB565 | Output interface format RGB565. |
| GLCDC_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB | Output interface format Serial RGB. |

### ◆ glcdc_dithering_output_format_t

| enum glcdc_dithering_output_format_t | |
|---|---|
| Dithering output format | |
| Enumerator | |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB888 | Dithering output format RGB888. |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB666 | Dithering output format RGB666. |
| GLCDC_DITHERING_OUTPUT_FORMAT_RGB565 | Dithering output format RGB565. |

## Function Documentation

### ◆ R_GLCDC_Open()

fsp_err_t R_GLCDC_Open ( display_ctrl_t *const *p_api_ctrl*, display_cfg_t const *const *p_cfg* )

Open GLCDC module. Implements display_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ALREADY_OPEN | Device was already open. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_CLOCK_GENERATION | Dot clock cannot be generated from clock source. |
| FSP_ERR_INVALID_TIMING_SETTING | Invalid panel timing parameter. |
| FSP_ERR_INVALID_LAYER_SETTING | Invalid layer setting found. |
| FSP_ERR_INVALID_ALIGNMENT | Input buffer alignment invalid. |
| FSP_ERR_INVALID_GAMMA_SETTING | Invalid gamma correction setting found |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid brightness correction setting found |

***Note***

*PCLKA must be supplied to Graphics LCD Controller (GLCDC) and GLCDC pins must be set in IOPORT before calling this API.*

### ◆ R_GLCDC_Close()

fsp_err_t R_GLCDC_Close ( display_ctrl_t *const *p_api_ctrl*)

Close GLCDC module. Implements display_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was closed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed when the GLCDC is updating register values internally. |

***Note***

*This API can be called when the driver is not in DISPLAY_STATE_CLOSED state. It returns an error if the register update operation for the background screen generation block is being held.*

### ◆ R_GLCDC_Start()

fsp_err_t R_GLCDC_Start ( display_ctrl_t *const *p_api_ctrl*)

Start GLCDC module. Implements display_api_t::start.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was started successfully. |
| FSP_ERR_NOT_OPEN | GLCDC module has not been opened. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |

***Note***

*This API can be called when the driver is not in DISPLAY_STATE_OPENED status.*

### ◆ R_GLCDC_Stop()

fsp_err_t R_GLCDC_Stop ( display_ctrl_t *const *p_api_ctrl*)

Stop GLCDC module. Implements display_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was stopped successfully |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL |
| FSP_ERR_INVALID_MODE | Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | The function call is performed while the GLCDC is updating register values internally. |

***Note***

*This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks, the graphics data I/F blocks, or the output control block is being held.*

### ◆ R_GLCDC_LayerChange()

fsp_err_t R_GLCDC_LayerChange ( display_ctrl_t const *const *p_api_ctrl*, display_runtime_cfg_t const *const *p_cfg*, display_frame_layer_t *layer* )

Change layer parameters of GLCDC module at runtime. Implements display_api_t::layerChange.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Changed layer parameters of GLCDC module successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_INVALID_MODE | A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC is updating register values internally. |

***Note***

*This API can be called when the driver is in DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.*

### ◆ R_GLCDC_BufferChange()

fsp_err_t R_GLCDC_BufferChange ( display_ctrl_t const *const *p_api_ctrl*, uint8_t *const *framebuffer*, display_frame_layer_t *layer* )

Change the framebuffer pointer for a layer. Implements display_api_t::bufferChange.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Changed layer parameters of GLCDC module successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_INVALID_MODE | A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_ALIGNMENT | The framebuffer pointer is not 64-byte aligned. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC is updating register values internally. |

***Note***

*This API can be called when the driver is in DISPLAY_STATE_OPENED state or higher. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.*

### ◆ R_GLCDC_ColorCorrection()

fsp_err_t R_GLCDC_ColorCorrection ( display_ctrl_t const *const *p_api_ctrl*, display_correction_t const *const *p_correction* )

Perform color correction through the GLCDC module. Implements display_api_t::correction.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Color correction by GLCDC module was performed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the display correction structure is NULL. |
| FSP_ERR_INVALID_MODE | Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING. |
| FSP_ERR_INVALID_UPDATE_TIMING | A function call is performed while the GLCDC is updating registers internally. |
| FSP_ERR_UNSUPPORTED | Feature not supported with the current configuration. |
| FSP_ERR_INVALID_BRIGHTNESS_SETTING | Invalid brightness correction setting found |

***Note***

*This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the output control block is being held.*

### ◆ R_GLCDC_ClutUpdate()

fsp_err_t R_GLCDC_ClutUpdate ( display_ctrl_t const *const *p_api_ctrl*, display_clut_cfg_t const *const *p_clut_cfg*, display_frame_layer_t *layer* )

Write an entire color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clut.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CLUT written successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or CLUT source data is NULL. |
| FSP_ERR_INVALID_UPDATE_TIMING | R_GLCDC_ClutEdit was already used to edit the specified CLUT this frame. |
| FSP_ERR_INVALID_CLUT_ACCESS | Illegal CLUT entry or size is specified. |

***Note***

*This API can be called any time. The written data will be used after the next vertical sync event.*

### ◆ R_GLCDC_ClutEdit()

fsp_err_t R_GLCDC_ClutEdit ( display_ctrl_t const *const *p_api_ctrl*, display_frame_layer_t *layer*, uint8_t *index*, uint32_t *color* )

Update an element of a color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clutEdit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CLUT element updated successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |

***Note***

*This API can be called any time. The written data will be used after the next vertical sync event.*

### ◆ R_GLCDC_ColorKeySet()

fsp_err_t R_GLCDC_ColorKeySet ( display_ctrl_t const *const *p_api_ctrl*, display_colorkeying_layer_t *key_cfg*, display_frame_layer_t *layer* )

Configuring color key is not supported for GLCDC.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | |

### ◆ R_GLCDC_StatusGet()

fsp_err_t R_GLCDC_StatusGet ( display_ctrl_t const *const *p_api_ctrl*, display_status_t *const *p_status* )

Get status of GLCDC module. Implements display_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got status successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the status structure is NULL. |

***Note***

*The GLCDC hardware starts the fading processing at the first Vsync after the previous LayerChange() call is held. Due to this behavior of the hardware, this API may not return DISPLAY_FADE_STATUS_FADING_UNDERWAY as the fading status, if it is called before the first Vsync after LayerChange() is called. In this case, the API returns DISPLAY_FADE_STATUS_PENDING, instead of DISPLAY_FADE_STATUS_NOT_UNDERWAY.*

### 5.2.8.6 JPEG Codec (r_jpeg)

Modules » Graphics

#### Functions

| | |
|---|---|
| fsp_err_t | R_JPEG_Open (jpeg_ctrl_t *const p_api_ctrl, jpeg_cfg_t const *const p_cfg) |
| fsp_err_t | R_JPEG_OutputBufferSet (jpeg_ctrl_t *p_api_ctrl, void *p_output_buffer, uint32_t output_buffer_size) |
| fsp_err_t | R_JPEG_InputBufferSet (jpeg_ctrl_t *const p_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size) |
| fsp_err_t | R_JPEG_DecodeLinesDecodedGet (jpeg_ctrl_t *p_api_ctrl, uint32_t *p_lines) |
| fsp_err_t | R_JPEG_DecodeImageSubsampleSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) |
| fsp_err_t | R_JPEG_DecodeHorizontalStrideSet (jpeg_ctrl_t *p_api_ctrl, uint32_t horizontal_stride) |
| fsp_err_t | R_JPEG_DecodeImageSizeGet (jpeg_ctrl_t *p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) |
| fsp_err_t | R_JPEG_DecodePixelFormatGet (jpeg_ctrl_t *p_api_ctrl, jpeg_color_space_t *p_color_space) |
| fsp_err_t | R_JPEG_EncodeImageSizeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_encode_image_size_t *p_image_size) |
| fsp_err_t | R_JPEG_ModeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_mode_t mode) |
| fsp_err_t | R_JPEG_Close (jpeg_ctrl_t *p_api_ctrl) |
| fsp_err_t | R_JPEG_StatusGet (jpeg_ctrl_t *p_api_ctrl, jpeg_status_t *p_status) |

#### Detailed Description

Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface.

# Overview

The JPEG Codec is a hardware block providing accelerated JPEG image encode and decode functionality independent of the CPU. Images can optionally be partially processed facilitating streaming applications.

### Features

The JPEG Codec provides a number of options useful in a variety of applications:

- Basic encoding and decoding
- Streaming input and/or output
- Decoding JPEGs of unknown size
- Shrink (sub-sample) an image during the decoding process
- Rearrange input and output byte order (byte, word and/or longword swap)
- JPEG error detection

The specifications for the codec are as follows:

| Feature | Options |
|---|---|
| Decompression input formats | Baseline JPEG Y'CbCr 4:4:4, 4:2:2, 4:2:0 and 4:1:1 |
| Decompression output formats | ARGB8888, RGB565 |
| Compression input formats | Raw Y'CbCr 4:2:2 only |
| Compression output formats | Baseline JPEG Y'CbCr 4:2:2 only |
| Byte reordering | Byte, halfword and/or word swapping on input and output |
| Interrupt sources | Image size acquired, input/output data pause, decode complete, error |
| Compatible image sizes | See Minimum Coded Unit (MCU) below |

# Configuration

### Build Time Configurations for r_jpeg

The following build time configurations are defined in fsp_cfg/r_jpeg_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected, code for parameter checking is included in the build. |
| Decode Support | • Enabled<br>• Disabled | Enabled | If selected, code for decoding JPEG images is included in the build. |
| Encode Support | • Enabled<br>• Disabled | Disabled | If selected, code for encoding JPEG images is included in the build. |

### Configurations for Graphics > JPEG Codec (r_jpeg)

This module can be added to the Stacks tab via New Stack > Graphics > JPEG Codec (r_jpeg).

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_jpeg0 | Module name. |
| Default mode | • Decode<br>• Encode | Decode | Set the mode to use when calling R_JPEG_Open. This parameter is only used when both Encode and Decode support are enabled. |
| Decode | | | |
| Input byte order | MCU Specific Options | | Select the byte order of the input data for decoding. |
| Output byte order | MCU Specific Options | | Select the byte order of the output data for decoding. |
| Output color format | • ARGB8888 (32-bit)<br>• RGB565 (16-bit) | RGB565 (16-bit) | Select the output pixel format for decode operations. |
| Output alpha (ARGB8888 only) | Value must be an 8-bit integer (0-255) | 255 | Specify the alpha value to apply to each output pixel when ARGB8888 format is chosen. |
| Callback | Name must be a valid C symbol | NULL | If a callback function is provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers. |
| Encode | | | |
| Horizontal resolution | Value cannot be greater than 65535 and must be a non-negative integer divisible by 16 | 480 | Horizontal resolution of the raw image (in pixels). This value can be configured at runtime via R_JPEG_ImageSizeSet. |
| Vertical resolution | Value cannot be greater than 65535 and must be a non-negative integer divisible by 8 | 272 | Vertical resolution of the raw image. This value can be configured at runtime via R_JPEG_ImageSizeSet. |
| Horizontal stride | Value cannot be greater than 65535 and must be a non-negative integer | 480 | Horizontal stride of the raw image buffer (in pixels). This value can be configured at |

| | | | |
|---|---|---|---|
| | | | runtime via R_JPEG_ImageSizeSet. |
| Input byte order | MCU Specific Options | | Select the byte order of the input data for encoding. |
| Output byte order | MCU Specific Options | | Select the byte order of the output data for encoding. |
| Reset interval | Value cannot be greater than 65535 and must be a non-negative integer | 512 | Set the number of MCUs between RST markers. A value of 0 will disable DRI and RST marker output. |
| Quality factor | Value must be between 1 and 100 and must be an integer | 50 | Set the quality factor for encoding (1-100). Lower values produce smaller images at the cost of image quality. |
| Callback | Name must be a valid C symbol | NULL | If a callback function is provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers. |
| Interrupts | | | |
| Decode Process Interrupt Priority | MCU Specific Options | | Select the decompression interrupt priority. |
| Data Transfer Interrupt Priority | MCU Specific Options | | Select the data transfer interrupt priority. |

### Clock Configuration

The peripheral clock for this module is PCLKA. No clocks are provided by this module.

### Pin Configuration

This module does not have any input or output pin connections.

# Usage Notes

### Overview

The JPEG Codec contains both decode and encode hardware. While these two functions are largely independent in configuration only one can be used at a time.

To switch from decode to encode mode (or vice versa) use R_JPEG_ModeSet while the JPEG Codec is idle.

### Status

The status value (jpeg_status_t) provided by the callback and by R_JPEG_StatusGet is a bitfield that encompasses all potential status indication conditions. One or more statuses can be set simultaneously.

### Decoding Process

JPEG decoding can be performed in several ways depending on the application:

- To perform the simplest decode operation where all dimensions are known:
  - Set the input buffer, stride and output buffer then wait for a callback with status JPEG_STATUS_OPERATION_COMPLETE.
- To pause after decoding the JPEG header (in order to acquire image dimensions and secure an output buffer):
  - Call R_JPEG_InputBufferSet before setting the output buffer and wait for a callback with status JPEG_STATUS_IMAGE_SIZE_READY.
- To decode a partial JPEG image then pause until the next chunk is available:
  - Specify a size smaller than the full JPEG data when calling R_JPEG_InputBufferSet.
- To pause decoding once an output buffer is filled:
  - Specify a size smaller than the full decoded image when calling R_JPEG_OutputBufferSet.

The flowchart below illustrates the steps necessary to handle any decode operation. The statuses given in blue are part of jpeg_status_t with the JPEG_DECODE_STATUS prefix omitted.

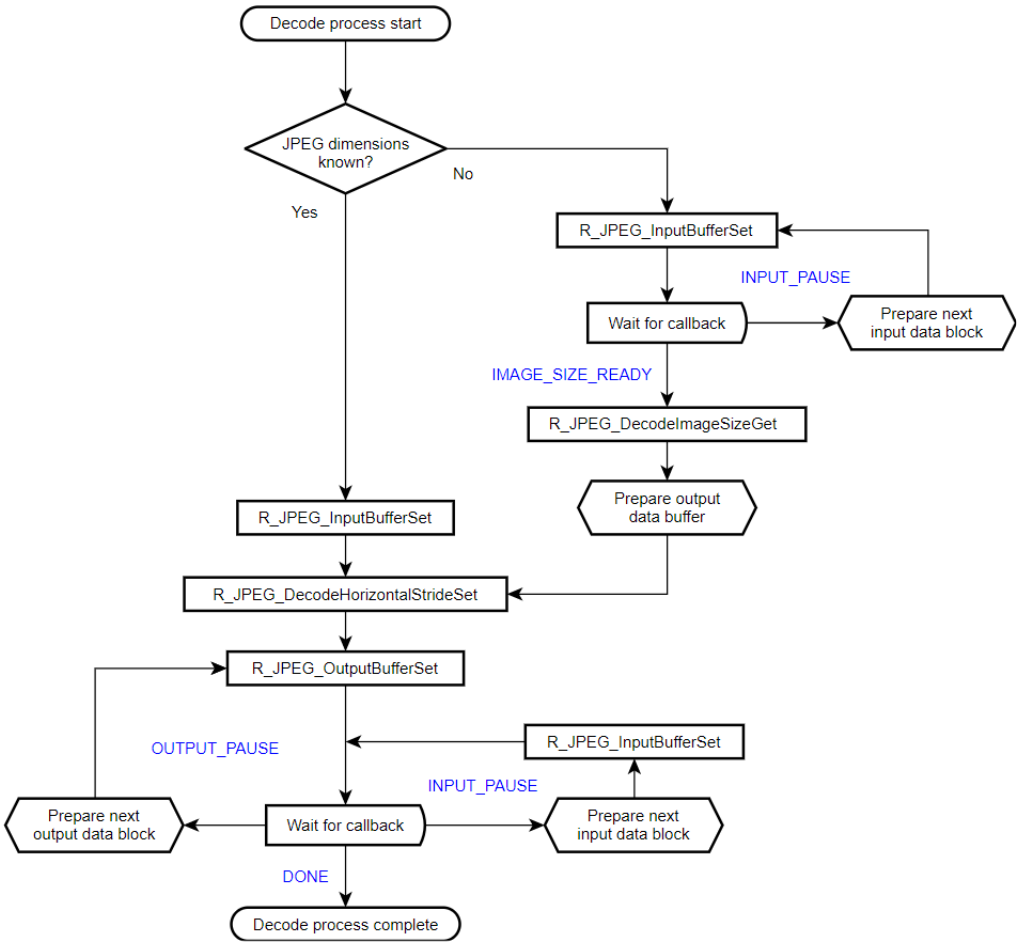



Figure 256: JPEG Decode Operational Flow

### Encoding Process

As compared to decoding, encoding is fairly straightforward. The only option available is to stream input data if desired. The flowchart below details the steps needed to compress an image.

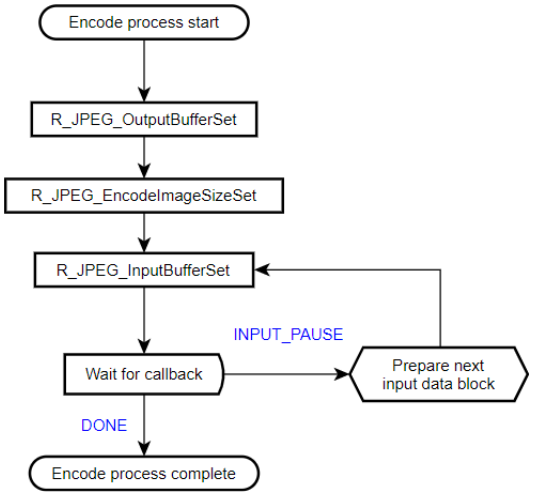



Figure 257: JPEG Encode Operational Flow

### Handling Failed Operations

If an encode or decode operation fails or times out while the codec is running, the peripheral must be reset before it is used again. To reset the JPEG Codec simply close and re-open the module by calling R_JPEG_Close followed by R_JPEG_Open.

### Limitations

Developers should be aware of the following limitations when using the JPEG API.

### Minimum Coded Unit (MCU)

The JPEG Codec can only correctly process images that are an even increment of minimum coded units (MCUs). In other words, depending on the format the width and height of an image to be encoded or decoded must be divisible by the following:

| Format | Horizontal | Vertical |
|---|---|---|
| Y'CbCr 4:4:4 | 8 pixels | 8 lines |
| Y'CbCr 4:2:2 | 16 pixels | 8 lines |
| Y'CbCr 4:1:1 | 32 pixels | 8 lines |
| Y'CbCr 4:2:0 | 16 pixels | 16 lines |

*Note*

*Because encoding is limited to Y'CbCr 4:2:2, raw pixel input data must always be in whole increments of 16x8 pixels.*

### Encoding Input Format

The encoding unit only supports Y'CbCr 4:2:2 input. Raw RGB888 data can be converted to this format as follows:

```
y =       (0.299000f * r) + (0.587000f * g) + (0.114000f * b);
cb = 128 - (0.168736f * r) - (0.331264f * g) + (0.500000f * b);
cr = 128 + (0.500000f * r) - (0.418688f * g) - (0.081312f * b);
```

While these equations are mathematically simple they do use the floating-point unit. To speed things up we can multiply the coefficients by 256 and divide the sum by 256...

```
y =       ((76.5440f * r) + (150.272f * g) + (29.1840f * b)) / 256;
cb = 128 - ((43.1964f * r) - (84.8036f * g) + (128.000f * b)) / 256;
cr = 128 + ((128.000f * r) - (107.184f * g) - (20.8159f * b)) / 256;
```

...which allows the formulas to be calculated entirely with shifts and addition (coefficients rounded to the nearest integer):

```
y =       (   (r << 6) + (r << 3) + (r << 2) + r
            + (g << 7) + (g << 4) + (g << 2) + (g << 1)
            + (b << 4) + (b << 3) + (b << 2) + b
           ) >> 8;
cb = 128 - (   (r << 5) + (r << 3) + (r << 1) + r
            + (g << 6) + (g << 4) + (g << 2) + g
            - (b << 7)
           ) >> 8;
cr = 128 + (   (r << 7)
            - (g << 6) - (g << 5) - (g << 3) - (g << 1) - g
            - (b << 4) - (b << 2) - b)
           ) >> 8;
```

To compose the final Y'CbCr 4:2:2 data the chroma of every two pixels must be averaged. **In addition, the JPEG Codec expects chrominance values to be in the range -127..127 instead of the standard 1..255.**

```
cb = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - 128);
cr = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - 128);
```

Finally, the below equation composes two 4:2:2 output pixels at a time with standard byte order (JPEG_DATA_ORDER_NORMAL):

```
   out = y0 + (cb << 8) + (y1 << 16) + (cr << 24);
```

*Note*

*RGB565 pixels must be upscaled to RGB888 before using the above formulas. Refer to the below example on Y'CbCr Conversion for implementation details.*

# Examples

### Basic Decode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and decode an image.

```
void jpeg_decode_basic (void)
{
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, JPEG_PTR, JPEG_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Set horizontal stride of output buffer */
    err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, JPEG_HSIZE);
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, decode_buffer, sizeof(decode_buffer));
    assert(FSP_SUCCESS == err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & (JPEG_STATUS_OPERATION_COMPLETE | JPEG_STATUS_ERROR)))
    {
```

```
       err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
}
```

### Streaming Input/Output Example

In this example JPEG data is read in 512-byte chunks. Decoding is paused when a chunk is read and once the JPEG header is decoded. The image is decoded 16 lines at a time.

*Note*

*Streaming is always bypassed when a given buffer's size encompasses the entire input or output image, respectively. Though this example decodes via smaller chunks the input and output data are still contiguous for ease of demonstration. Refer to the comments for further insight as to how to implement streaming with different JPEG/output buffer size combinations.*

```
#define JPEG_INPUT_SIZE_BYTES 512U
/* JPEG Codec status */
static volatile jpeg_status_t g_jpeg_status = JPEG_STATUS_NONE;
/* JPEG event flag */
static volatile uint8_t jpeg_event = 0;
/* Callback function for JPEG decode interrupts */
void jpeg_decode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
/* Simple wait that returns 1 if no event happened within the timeout period */
static uint8_t jpeg_event_wait (void)
{
    uint32_t timeout_timer = JPEG_EVENT_TIMEOUT;
 while (!jpeg_event && --timeout_timer)
    {
 /* Spin here until an event callback or timeout */
    }
    jpeg_event = 0;
```

```
 return timeout_timer ? 0 : 1;
}
/* Decode a JPEG image to a buffer using streaming input and output */
void jpeg_decode_streaming (void)
{
    uint8_t    * p_jpeg = (uint8_t *) JPEG_PTR;
 jpeg_status_t status = (jpeg_status_t) 0;
    uint8_t       timeout = 0;
 fsp_err_t    err;
 /* Number of input bytes to read at a time */
    uint32_t input_bytes = JPEG_INPUT_SIZE_BYTES;
 /* Open JPEG unit and start decode */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(status & JPEG_STATUS_ERROR) && !timeout)
    {
 /* Set the input buffer to read `input_bytes` bytes at a time */
       err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
     assert(FSP_SUCCESS == err);
 /* This delay is required for streaming input mode to function correctly.
  * (Without this delay the JPEG Codec will not correctly locate markers in the file
header.) */
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MICROSECONDS);
 /* Wait for a callback */
       timeout = jpeg_event_wait();
 /* Get the status from the callback */
       status = g_jpeg_status;
 /* Break if the header has finished decoding */
 if (status & JPEG_STATUS_IMAGE_SIZE_READY)
     {
 break;
     }
 /* Move pointer to next block of input data (if needed) */
```

```
      p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
   }
/* Get image size */
   uint16_t horizontal;
   uint16_t vertical;
   err = R_JPEG_DecodeImageSizeGet(&g_jpeg_ctrl, &horizontal, &vertical);
   assert(FSP_SUCCESS == err);
/* Prepare output data buffer here if needed (already allocated in this example) */
   uint8_t * p_output = decode_buffer;
/* Set horizontal stride */
   err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, horizontal);
   assert(FSP_SUCCESS == err);
/* Calculate the number of bytes that will fit in the buffer (16 lines in this
example) */
   uint32_t output_size = horizontal * 16U * 4U;
/* Start decoding by setting the output buffer */
   err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
   assert(FSP_SUCCESS == err);
while (!(status & JPEG_STATUS_ERROR) && !timeout)
   {
/* Wait for a callback */
      timeout = jpeg_event_wait();
/* Get the status from the callback */
      status = g_jpeg_status;
/* Break if decoding is complete */
if (status & JPEG_STATUS_OPERATION_COMPLETE)
     {
break;
     }
if (status & JPEG_STATUS_OUTPUT_PAUSE)
     {
/* Draw the JPEG work buffer to the framebuffer here (if needed) */
/* Move pointer to next block of output data (if needed) */
           p_output += output_size;
```

```
 /* Set the output buffer to the next 16-line block */
      err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
      assert(FSP_SUCCESS == err);
      }
 if (status & JPEG_STATUS_INPUT_PAUSE)
      {
 /* Get next block of input data */
         p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
 /* Set the new input buffer pointer */
         err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
      assert(FSP_SUCCESS == err);
      }
    }
 /* Close driver to allow encode operations if needed */
    err = R_JPEG_Close(&g_jpeg_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Encode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and encode an image.

*Note*

*This example assumes image dimensions are provided in the configuration. If this is not the case, R_JPEG_EncodeImageSizeSet must be used to set the size before calling R_JPEG_InputBufferSet.*

```
void jpeg_encode_basic (void)
{
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
    assert(FSP_SUCCESS == err);
```

```
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, RAW_YCBCR_IMAGE_PTR, IMAGE_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & JPEG_STATUS_OPERATION_COMPLETE))
    {
       err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
}
```

### Streaming Encode Example

In this example the raw input data is provided in smaller chunks. This can help significantly reduce buffer size and improve throughput when streaming in raw data from an outside source.

```
/* Callback function for JPEG encode interrupts */
void jpeg_encode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
void jpeg_encode_streaming (void)
{
    uint8_t   timeout = 0;
    uint8_t * p_chunk = (uint8_t *) RAW_YCBCR_IMAGE_PTR;
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
```

```
   err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
   assert(FSP_SUCCESS == err);
/* Set the image size */
jpeg_encode_image_size_t image_size;
   image_size.horizontal_resolution    = X_RESOLUTION;
   image_size.vertical_resolution      = Y_RESOLUTION;
   image_size.horizontal_stride_pixels = H_STRIDE;
   err = R_JPEG_EncodeImageSizeSet(&g_jpeg_ctrl, &image_size);
   assert(FSP_SUCCESS == err);
/* Calculate the size of the input data chunk (16 lines in this example) */
   uint32_t chunk_size = H_STRIDE * 16U * YCBCR_BYTES_PER_PIXEL;
while (!timeout)
   {
/* Set the input buffer */
      err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_chunk, chunk_size);
     assert(FSP_SUCCESS == err);
/* Wait for a callback */
      timeout = jpeg_event_wait();
if (g_jpeg_status & JPEG_STATUS_OPERATION_COMPLETE)
     {
/* Encode complete */
break;
     }
if (g_jpeg_status & JPEG_STATUS_INPUT_PAUSE)
     {
/* Load next block of input data here (if needed) */
           p_chunk += chunk_size;
     }
   }
}
```

### Y'CbCr Conversion

The below function is provided as a reference for how to convert RGB values to Y'CbCr for use with the JPEG Codec.

*Note*

*This function is only partially optimized for clarity. Further appllication-specific size- or speed-based optimizations should be considered when implementing in an actual project.*

```
#define RGB565_G_MASK 0x07E0
#define RGB565_B_MASK 0x001F
#define C_0 128
typedef enum e_pixel_format
{
    PIXEL_FORMAT_ARGB8888,
    PIXEL_FORMAT_RGB565
} pixel_format_t;
/* 5-bit to 8-bit LUT */
const uint8_t lut_32[] =
{
    0,   8,   16, 25, 33, 41, 49, 58,
    66, 74, 82, 90, 99, 107, 115, 123,
    132, 140, 148, 156, 165, 173, 181, 189,
    197, 206, 214, 222, 230, 239, 247, 255
};
/* 6-bit to 8-bit LUT */
const uint8_t lut_64[] =
{
    0,   4,   8,   12, 16, 20, 24, 28,
    32, 36, 40, 45, 49, 53, 57, 61,
    65, 69, 73, 77, 81, 85, 89, 93,
    97, 101, 105, 109, 113, 117, 121, 125,
    130, 134, 138, 142, 146, 150, 154, 158,
    162, 166, 170, 174, 178, 182, 186, 190,
    194, 198, 202, 206, 210, 215, 219, 223,
    227, 231, 235, 239, 243, 247, 251, 255
};
void bitmap_rgb2ycbcr(uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format);
/*********************************************************************************
**********************************
```

```
 * Convert an RGB buffer to Y'CbCr 4:2:2.
 *
 * NOTE: The width (in pixels) of the image to be converted must be divisible by 2.
 *
 * Parameters:
 * out Pointer to output buffer
 * in Pointer to input buffer
 * len Length of input buffer (in pixels)
 * format Input buffer format (ARGB8888 or RGB565)
 ***********************************************************************************
*********************************/
void bitmap_rgb2ycbcr (uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format)
{
    uint16_t in0;
    uint16_t in1;
    uint32_t r0;
    uint32_t g0;
    uint32_t b0;
    uint32_t r1;
    uint32_t g1;
    uint32_t b1;
    uint8_t y0;
    uint8_t y1;
    uint8_t cb0;
    uint8_t cr0;
    uint8_t cb1;
    uint8_t cr1;
 /* Divide length by 2 as we're working with two pixels at a time */
    len >>= 1;
 /* Perform the conversion */
 while (len)
    {
 /* Get R, G and B channel values */
```

```
if (format == PIXEL_FORMAT_RGB565)
     {
/* Get next two 16-bit values */
           in0 = *((uint16_t *) in);
           in += 2;
           in1 = *((uint16_t *) in);
           in += 2;
/* Decompose into individual channels */
           r0 = in0 >> 11;
           g0 = (in0 & RGB565_G_MASK) >> 5;
           b0 = in0 & RGB565_B_MASK;
           r1 = in1 >> 11;
           g1 = (in1 & RGB565_G_MASK) >> 5;
           b1 = in1 & RGB565_B_MASK;
     }
else
     {
/* Get each ARGB8888 channel in sequence, skipping alpha */
           b0 = *in++;
           g0 = *in++;
           r0 = *in++;
           in++;
           b1 = *in++;
           g1 = *in++;
           r1 = *in++;
           in++;
     }
/* Convert RGB565 data to RGB888 */
if (PIXEL_FORMAT_RGB565 == format)
     {
           r0 = lut_32[r0];
           g0 = lut_64[g0];
           b0 = lut_32[b0];
           r1 = lut_32[r1];
```

```
            g1 = lut_64[g1];
            b1 = lut_32[b1];
        }
 /* Calculate Y'CbCr 4:4:4 values for the two pixels */
 /* Algorithm based on method shown here: https://sistenix.com/rgb2ycbcr.html */
 /* Original coefficients from https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion */
       y0 = (uint8_t) (((r0 << 6) + (r0 << 3) + (r0 << 2) + r0 +
                        (g0 << 7) + (g0 << 4) + (g0 << 2) + (g0 << 1) +
                        (b0 << 4) + (b0 << 3) + (b0 << 2) + b0
                        ) >> 8);
       cb0 = (uint8_t) (C_0 - (((r0 << 5) + (r0 << 3) + (r0 << 1) + r0 +
                                (g0 << 6) + (g0 << 4) + (g0 << 2) + g0 -
                                (b0 << 7)
                                ) >> 8));
       cr0 = (uint8_t) (C_0 + (((r0 << 7) -
                                (g0 << 6) - (g0 << 5) - (g0 << 3) - (g0 << 1) - g0 -
                                (b0 << 4) - (b0 << 2) - b0
                                ) >> 8));
       y1 = (uint8_t) (((r1 << 6) + (r1 << 3) + (r1 << 2) + r1 +
                        (g1 << 7) + (g1 << 4) + (g1 << 2) + (g1 << 1) +
                        (b1 << 4) + (b1 << 3) + (b1 << 2) + b1
                        ) >> 8);
       cb1 = (uint8_t) (C_0 - (((r1 << 5) + (r1 << 3) + (r1 << 1) + r1 +
                                (g1 << 6) + (g1 << 4) + (g1 << 2) + g1 -
                                (b1 << 7)
                                ) >> 8));
       cr1 = (uint8_t) (C_0 + (((r1 << 7) -
                                (g1 << 6) - (g1 << 5) - (g1 << 3) - (g1 << 1) - g1 -
                                (b1 << 4) - (b1 << 2) - b1
                                ) >> 8));
 /* The above code is based on the floating point method shown here: */
 // y0 = (uint8_t) ((0.299F * (float) r0) + (0.587F * (float) g0) + (0.114F * (float)
b0));
 // y1 = (uint8_t) ((0.299F * (float) r1) + (0.587F * (float) g1) + (0.114F * (float)
```

```
b1));
 // cb0 = (uint8_t) (128.0F - (0.168736F * (float) r0) - (0.331264F * (float) g0) +
(0.5F * (float) b0));
 // cb1 = (uint8_t) (128.0F - (0.168736F * (float) r1) - (0.331264F * (float) g1) +
(0.5F * (float) b1));
 // cr0 = (uint8_t) (128.0F + (0.5F * (float) r0) - (0.418688F * (float) g0) -
(0.081312F * (float) b0));
 // cr1 = (uint8_t) (128.0F + (0.5F * (float) r1) - (0.418688F * (float) g1) -
(0.081312F * (float) b1));
 /* NOTE: The JPEG Codec expects signed instead of unsigned chrominance values. */
 /* Convert chrominance to -127..127 instead of 1..255 */
       cb0 = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - C_0);
       cr0 = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - C_0);
 /* Convert the two 4:4:4 values into 4:2:2 by averaging the chroma, then write to
output */
       *out++ = (uint32_t) (y0 + (cb0 << 8) + (y1 << 16) + (cr0 << 24));
       len--;
    }
}
```

## Data Structures

| | |
|---|---|
| struct | jpeg_instance_ctrl_t |

## Data Structure Documentation

### ◆ jpeg_instance_ctrl_t

| struct jpeg_instance_ctrl_t | | |
|---|---|---|
| JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | JPEG Codec driver status. |
| jpeg_status_t | status | JPEG Codec operational status. |
| fsp_err_t | error_code | JPEG Codec error code (if any). |
| jpeg_mode_t | mode | Current mode (decode or encode). |
| uint32_t | horizontal_stride_bytes | Horizontal Stride settings. |

| | | |
|---|---|---|
| uint32_t | output_buffer_size | Output buffer size. |
| jpeg_cfg_t const * | p_cfg | JPEG Decode configuration struct. |
| void const * | p_extend | JPEG Codec hardware dependent configuration */. |
| jpeg_decode_pixel_format_t | pixel_format | Pixel format. |
| uint16_t | total_lines_decoded | Track the number of lines decoded so far. |
| jpeg_decode_subsample_t | horizontal_subsample | Horizontal sub-sample setting. |
| uint16_t | lines_to_encode | Number of lines to encode. |
| uint16_t | vertical_resolution | vertical size |
| uint16_t | total_lines_encoded | Number of lines encoded. |

## Function Documentation

### ◆ R_JPEG_Open()

fsp_err_t R_JPEG_Open ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_cfg_t const *const *p_cfg* )

Initialize the JPEG Codec module.

***Note***

*This function configures the JPEG Codec for operation and sets up the registers for data format and pixel format based on user-supplied configuration parameters. Interrupts are enabled to support callbacks.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | JPEG Codec module is properly configured and is ready to take input data. |
| FSP_ERR_ALREADY_OPEN | JPEG Codec is already open. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_IRQ_BSP_DISABLED | JEDI interrupt does not have an IRQ number. |
| FSP_ERR_INVALID_ARGUMENT | (Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid. |
| FSP_ERR_INVALID_ALIGNMENT | (Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes. |

### ◆ R_JPEG_OutputBufferSet()

fsp_err_t R_JPEG_OutputBufferSet ( jpeg_ctrl_t * *p_api_ctrl*, void * *p_output_buffer*, uint32_t *output_buffer_size* )

Assign a buffer to the JPEG Codec for storing output data.

***Note***

*In Decode mode, the number of image lines to be decoded depends on the size of the buffer and the horizontal stride settings. Once the output buffer size is known, the horizontal stride value is known, and the input pixel format is known (the input pixel format is obtained by the JPEG decoder from the JPEG headers), the driver automatically computes the number of lines that can be decoded into the output buffer. After these lines are decoded, the JPEG engine pauses and a callback function is triggered, so the application is able to provide the next buffer for the JPEG module to resume the operation.*

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set and the output buffer is big enough to hold at least eight lines of decoded image data.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The output buffer is properly assigned to JPEG codec device. |
| FSP_ERR_ASSERTION | Pointer to the control block or output_buffer is NULL or output_buffer_size is 0. |
| FSP_ERR_INVALID_ALIGNMENT | Buffer starting address is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE | The number of horizontal pixels exceeds horizontal memory stride. |
| FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH | Invalid buffer size. |
| FSP_ERR_IN_USE | The output buffer cannot be changed during codec operation. |

### ◆ R_JPEG_InputBufferSet()

fsp_err_t R_JPEG_InputBufferSet ( jpeg_ctrl_t *const *p_api_ctrl*, void * *p_data_buffer*, uint32_t *data_buffer_size* )

Assign an input data buffer to the JPEG codec for processing.

***Note***

*After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_PRV_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so processing can resume.*
*The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least one line of decoded image data.*

If zero is provided for the decode data buffer size the JPEG Codec will never pause for more input data and will continue to read until either an image has been fully decoded or an error condition occurs.

***Note***

*When encoding images the minimum data buffer size is 8 lines by 16 Y'CbCr 4:2:2 pixels (256 bytes). This corresponds to one minimum coded unit (MCU) of the resulting JPEG output.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The input data buffer is properly assigned to JPEG Codec device. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL, or the pointer to the input_buffer is NULL, or the input_buffer_size is 0. |
| FSP_ERR_INVALID_ALIGNMENT | Buffer starting address is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_IN_USE | The input buffer cannot be changed while the codec is running. |
| FSP_ERR_INVALID_CALL | In encode mode the output buffer must be set first. |
| FSP_ERR_JPEG_IMAGE_SIZE_ERROR | The buffer size is smaller than the minimum coded unit (MCU). |

### ◆ R_JPEG_DecodeLinesDecodedGet()

fsp_err_t R_JPEG_DecodeLinesDecodedGet ( jpeg_ctrl_t * *p_api_ctrl*, uint32_t * *p_lines* )

Returns the number of lines decoded into the output buffer.

***Note***

*Use this function to retrieve the number of image lines written to the output buffer after a partial decode operation. Combined with the horizontal stride settings and the output pixel format the application can compute the amount of data to read from the output buffer.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Line count successfully returned. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_lines is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodeImageSubsampleSet()

fsp_err_t R_JPEG_DecodeImageSubsampleSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_decode_subsample_t *horizontal_subsample*, jpeg_decode_subsample_t *vertical_subsample* )

Configure horizontal and vertical subsampling.

***Note***

*This function can be used to scale the output of decoded image data.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Horizontal subsample value is properly configured. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodeHorizontalStrideSet()

fsp_err_t R_JPEG_DecodeHorizontalStrideSet ( jpeg_ctrl_t * *p_api_ctrl*, uint32_t *horizontal_stride* )

Configure horizontal stride setting for decode operations.

***Note***

*If the image size is known prior to the open call and/or the output buffer stride is constant, pass the horizontal stride value in the jpeg_cfg_t structure. Otherwise, after the image size becomes available use this function to set the output buffer horizontal stride value.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Horizontal stride value is properly configured. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_INVALID_ALIGNMENT | Horizontal stride is zero or is not 8-byte aligned. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_DecodeImageSizeGet()

fsp_err_t R_JPEG_DecodeImageSizeGet ( jpeg_ctrl_t * *p_api_ctrl*, uint16_t * *p_horizontal_size*, uint16_t * *p_vertical_size* )

Obtain the size of an image being decoded.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The image size is available and the horizontal and vertical values are stored in the memory pointed to by p_horizontal_size and p_vertical_size. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL and/or size is not ready. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

### ◆ R_JPEG_DecodePixelFormatGet()

fsp_err_t R_JPEG_DecodePixelFormatGet ( jpeg_ctrl_t * *p_api_ctrl*, jpeg_color_space_t * *p_color_space* )

Get the color format of the JPEG being decoded.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The color format was successfully retrieved. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

### ◆ R_JPEG_EncodeImageSizeSet()

fsp_err_t R_JPEG_EncodeImageSizeSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_encode_image_size_t * *p_image_size* )

Set the image dimensions for an encode operation.

***Note***

*Image dimensions must be set before setting the input buffer.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Image size was successfully written to the JPEG Codec. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_image_size is NULL. |
| FSP_ERR_INVALID_ALIGNMENT | Horizontal stride is not 8-byte aligned. |
| FSP_ERR_INVALID_ARGUMENT | Horizontal or vertical resolution is invalid or zero. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |
| FSP_ERR_IN_USE | Image parameters cannot be changed while the codec is running. |

### ◆ R_JPEG_ModeSet()

fsp_err_t R_JPEG_ModeSet ( jpeg_ctrl_t *const *p_api_ctrl*, jpeg_mode_t *mode* )

Switch between encode and decode mode (or vice-versa).

***Note***

*The codec must not be idle in order to switch modes.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Mode changed successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | JPEG Codec is currently in use. |
| FSP_ERR_INVALID_ARGUMENT | (Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid. |
| FSP_ERR_INVALID_ALIGNMENT | (Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes. |

### ◆ R_JPEG_Close()

fsp_err_t R_JPEG_Close ( jpeg_ctrl_t * *p_api_ctrl*)

Cancel an outstanding JPEG codec operation and close the device.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The JPEG unit is stopped and the driver is closed. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | JPEG not opened. |

### ◆ R_JPEG_StatusGet()

| fsp_err_t R_JPEG_StatusGet ( jpeg_ctrl_t * *p_api_ctrl*, jpeg_status_t * *p_status* ) |
|---|
| Get the status of the JPEG codec. This function can also be used to poll the device. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The status information is successfully retrieved. |
| FSP_ERR_ASSERTION | Pointer to the control block or p_status is NULL. |
| FSP_ERR_NOT_OPEN | JPEG is not opened. |

#### 5.2.8.7 LVGL Port (rm_lvgl_port)

Modules » Graphics

**Functions**

| | |
|---|---|
| fsp_err_t | RM_LVGL_PORT_Open (rm_lvgl_port_instance_ctrl_t *const p_instance_ctrl, rm_lvgl_port_cfg_t const *const p_cfg) |

**Detailed Description**

LVGL graphics library port for RA MCUs.

# Overview

The LVGL RA Port module provides the configuration necessary for use of the LVGL graphics library on RA products. The port provides integration for GLCDC into the LVGL display system, a time source for LVGL timer logic, and optional integration with FreeRTOS.

*Note*

*This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. Please consult the LVGL Project Documentation for details on how to use LVGL in your project.*

### Hardware Acceleration

Hardware acceleration is optionally provided by DRW using a lightly modified fork of the upstream driver provided by the LVGL project. Most rendering operations are supported.

# Configuration

### Build Time Configurations for lvgl

The following build time configurations are defined in fsp_cfg/lvgl/lvgl/lv_conf.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Custom lv_conf.h | Manual Entry | | Add a path to your custom lv_conf.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| LVGL | | | |
| LVGL > Display | | | |
| LV_COLOR_DEPTH | Value must equal 16 or 32. | 16 | Color depth. 16 or 32 supported. |
| LVGL > Memory | | | |
| LV_MEM_SIZE | Value must be an integer multiple of 1024 between 0x800 and 0xFFFFFFFF. | 0x10000 | Size of the memory available for `lv_malloc()` in bytes. |
| LV_DRAW_LAYER_SIMPLE_BUF_SIZE | Value must be an integer multiple of 1024 between 0x400 and 0xFFFFFFFF. | 0x6000 | Buffer size for simple layer chunks. |
| LV_DRAW_LAYER_MAX_MEMORY | Value must be an integer. | 0 | Limit the max allocated memory for simple and transformed layers.<br>0 for no limit.<br>Otherwise, should be greater than or equal to LV_DRAW_LAYER_SIMPLE_BUF_SIZE.<br>If transformed layers are used, it should be large enough for the largest widget (width x height x 4 area). |
| LV_DRAW_THREAD_STACK_SIZE | Value must be an integer multiple of 1024 between 0x400 and 0xFFFFFFFF. | 0x2000 | Stack size of the drawing thread in bytes. Only used when FreeRTOS integration is used. |
| LVGL > OS Integration | | | |
| LV_USE_OS | • LV_OS_NONE<br>• LV_OS_FREERTOS | LV_OS_FREERTOS | Select LVGL RTOS integration. |

| | | | |
|---|---|---|---|
| LV_DRAW_THREAD_PRIO | • LV_THREAD_PRIO_LOWEST<br>• LV_THREAD_PRIO_LOW<br>• LV_THREAD_PRIO_MID<br>• LV_THREAD_PRIO_HIGH<br>• LV_THREAD_PRIO_HIGHEST | LV_THREAD_PRIO_HIGH | Thread priority of the drawing task. |
| LVGL > Software Rendering | | | |
| LV_DRAW_SW_SUPPORT_RGB565 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_RGB565_SWAPPED | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_RGB565A8 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_RGB888 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_XRGB8888 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format. |

| | | | |
|---|---|---|---|
| | | | Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_ARGB8888 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_ARGB8888_PREMULTIPLIED | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_L8 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_AL88 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPORT_A8 | • Enabled<br>• Disabled | Enabled | Enable software rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LV_DRAW_SW_SUPPOR | • Enabled | Enabled | Enable software |

| | | | |
|---|---|---|---|
| T_I1 | • Disabled | | rendering support for the specified color format.<br>Disable to reduce code size.<br>Some formats may be used internally for some features. |
| LVGL > Hardware Rendering | | | |
| LV_USE_DRAW_DAVE2D | • Enabled<br>• Disabled | Enabled | Enable built-in DRW Engine support. |
| LVGL > Debug | | | |
| LV_ASSERT_HANDLER_INCLUDE | Manual Entry | <stdint.h> | Header file to include to support LV_ASSERT_HANDLER macro. |
| LV_ASSERT_HANDLER | Manual Entry | | Macro to execute when an LV_ASSERT occurs. |
| LVGL > Widgets | | | |
| LV_USE_OBJ_NAME | • Enabled<br>• Disabled | Disabled | Enable support for widget names. |
| LVGL > Input Devices | | | |
| LV_USE_GESTURE_RECOGNITION | • Enabled<br>• Disabled | Disabled | Enable support for multi-touch gesture recognition.<br>Requires floating point support. |
| LV_USE_IME_PINYIN | • Enabled<br>• Disabled | Disabled | Enable Pinyin input method. |
| LV_IME_PINYIN_USE_DEFAULT_DICT | • Enabled<br>• Disabled | Enabled | Use default thesaurus. |
| LV_IME_PINYIN_CAND_TEXT_NUM | Value must be positive. | 6 | Maximum number of displayed candidate panels. Adjust according to screen size. |
| LV_IME_PINYIN_USE_K9_MODE | • Enabled<br>• Disabled | Enabled | Use nine key (K9) input. |
| LV_IME_PINYIN_K9_CAND_TEXT_NUM | Value must be positive. | 3 | Maximum number of displayed candidate panels in K9 mode. Adjust according to screen size. |
| LVGL > Fonts | | | |
| LVGL > Fonts > Montserrat | | | |

| | | | |
|---|---|---|---|
| LV_FONT_MONTSERRAT_8 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 8. |
| LV_FONT_MONTSERRAT_10 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 10. |
| LV_FONT_MONTSERRAT_12 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 12. |
| LV_FONT_MONTSERRAT_14 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 14. |
| LV_FONT_MONTSERRAT_16 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 16. |
| LV_FONT_MONTSERRAT_18 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 18. |
| LV_FONT_MONTSERRAT_20 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 20. |
| LV_FONT_MONTSERRAT_22 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 22. |
| LV_FONT_MONTSERRAT_24 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 24. |
| LV_FONT_MONTSERRAT_26 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 26. |
| LV_FONT_MONTSERRAT_28 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 28. |
| LV_FONT_MONTSERRAT_30 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 30. |
| LV_FONT_MONTSERRAT_32 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 32. |
| LV_FONT_MONTSERRAT_34 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 34. |
| LV_FONT_MONTSERRAT_36 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 36. |

| | | | |
|---|---|---|---|
| LV_FONT_MONTSERRAT_38 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 38. |
| LV_FONT_MONTSERRAT_40 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 40. |
| LV_FONT_MONTSERRAT_42 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 42. |
| LV_FONT_MONTSERRAT_44 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 44. |
| LV_FONT_MONTSERRAT_46 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 46. |
| LV_FONT_MONTSERRAT_48 | • Enabled<br>• Disabled | Enabled | Enable built-in font (FontAwesome5, Montserrat) size 48. |
| LVGL > Fonts > DejaVu | | | |
| LV_FONT_DEJAVU_16_PERSIAN_HEBREW | • Enabled<br>• Disabled | Enabled | Enable built-in font DejaVu (Arabic, Hebrew, Persian) size 16. |
| LVGL > Fonts > SourceHanSansSC | | | |
| LV_FONT_SOURCE_HAN_SANS_SC_14_CJK | • Enabled<br>• Disabled | Enabled | Enable built-in font SourceHanSansSC size 14. |
| LV_FONT_SOURCE_HAN_SANS_SC_16_CJK | • Enabled<br>• Disabled | Enabled | Enable built-in font SourceHanSansSC size 16. |
| LVGL > Fonts > Unscii | | | |
| LV_FONT_UNSCII_8 | • Enabled<br>• Disabled | Enabled | Enable built-in font unscii-8 size 8. |
| LV_FONT_UNSCII_16 | • Enabled<br>• Disabled | Enabled | Enable built-in font unscii-8 size 16. |
| LV_FONT_DEFAULT | Name must be a valid C symbol. | lv_font_montserrat_14 | Set a default font for LVGL. Required. |
| LV_USE_FONT_PLACEHOLDER | • Enabled<br>• Disabled | Enabled | Draw a placeholder when a glyph is not found. |
| LV_USE_BIDI | • Enabled<br>• Disabled | Disabled | Enable bidirectional text. |
| LV_USE_ARABIC_PERSIAN_CHARS | • Enabled<br>• Disabled | Disabled | Enable Arabic and Persian language |

| | | | |
|---|---|---|---|
| | | | processing. |
| LV_USE_FONT_MANAGER | • Enabled<br>• Disabled | Disabled | Enable the font manager. |
| LV_FONT_MANAGER_NAME_MAX_LEN | Value must be positive. | 32 | Maximum font name length when font manager is enabled. |

**Build Time Configurations for rm_lvgl_port**

The following build time configurations are defined in fsp_cfg/middleware/rm_lvgl_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Provide Tick Callback | • Enabled<br>• Disabled | Enabled | Enable the LVGL port driver to provide a tick callback to LVGL. |

**Configurations for Graphics > LVGL Port (rm_lvgl_port)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_lvgl_port0 | Module name. |
| Target Display Layer | • Graphics Layer 1<br>• Graphics Layer 2 | Graphics Layer 1 | Specify which graphics screen to inherit the buffer and display dimensions from. |
| Callback Function | Must be a valid C symbol | NULL | If a callback function is provided it will be called when Display events occur. |

**Hardware Configuration**

No clocks or pins are directly required by this module. Please consult the GLCDC Usage Guide for its requirements here: Graphics LCD (r_glcdc)

**LVGL Configuration**

LVGL is configured via the use of a configuration header. The port layer provides configuration options within e2 Studio for commonly changed LVGL settings and autogenerates the required header files. It is also possible to provide a custom LVGL configuration header which overrides the LVGL configuration options set in e2 Studio, or configures options that are not exposed in the port layer in e2 Studio.

# Usage Notes

**Getting Started**

To get started with LVGL in an RA project the following must be performed:

1. In e2 studio, open the RA Configuration editor for your project
2. Add LVGL to your project in the Stacks view by clicking **New Stack -> Graphics -> LVGL**
    - If using FreeRTOS, first create a new thread and add the LVGL stack to it
3. Set the properties for the GLCDC module to match the timing and memory requirements of your display panel
4. Ensure interrupts on all modules are enabled:
    - GLCDC Vertical Position (GLCDC_VPOS) Interrupt
    - DRW Interrupt (if applicable)
5. Confirm stack size and global heap size are configured as needed
    - If using FreeRTOS, it is suggested to add the Heap_4 stack to provide memory allocation to DRW
    - Stack size must be adequate for the thread executing the LVGL timer handler loop
    - Stack size is configured per thread if FreeRTOS is used
    - Final heap and stack usage must be characterized
6. Ensure the configuration options for LVGL are set as necessary for your application
7. Confirm LVGL heap size is configured as needed
    - Final LVGL heap usage must be characterized
8. Confirm other LVGL memory parameters are configured as needed
    - Final memory usage for miscellaneous memory parameters must be characterized
9. Resolve any remaining constraints
10. Click Generate Project Content to save and commit configuration changes

At this point the project is now ready to build with LVGL. Please refer to the LVGL Project Documentation for details on how to use LVGL in your project, as well as LVGL demo and example code.

### Demos and Examples

The LVGL packed with FSP also includes the demos and examples directories that are in the LVGL project. To allow the demos to be included and built in a user project:

1. Create a custom LVGL configuration header, for example lv_conf_custom.h
2. Specify the header file name in the **General|Custom lv_conf.h** LVGL stack configuration property
3. Set the LV_BUILD_DEMOS macro to 1 in the custom header, to override the value set in the generated lv_conf.h
4. Set the desired LV_USE_DEMO_* macro to 1 in the the custom header, to enable the desired demo
5. Call the desired demo function prior to lv_timer_handler() to set up the demo

Including and building the examples in a user project are the same steps, but set the LV_BUILD_EXAMPLES macro to 1 in the custom header instead. No LV_USE_ macros should be required for the examples.

*Note*

*Not all upstream demos and examples material are available, so some demos or examples may not build or execute.*

### Libraries

The LVGL packed with FSP also includes the src/libs directory that is in the LVGL project. Only bin_decoder is a mandatory library used by LVGL. Other libraries are either first-party or third-party libraries available for use within LVGL.

*Note*

*The included libraries may not be applicable to an RA embedded context. The included libraries may not compile or function without further modification. Libraries like ThorVG require a C++ based FSP project. These libraries are included for convenience but are not validated or officially supported as part of the FSP LVGL integration. Not all libraries have been included.*

### Multi-thread Support

LVGL is not a thread safe library. Please refer to Tasks and threads for information on how to use LVGL in a multi-thread environment.

*Note*

*This includes both bare metal and FreeRTOS projects. RTOS support is optionally configured depending on whether FreeRTOS is used in the project.*

### Limitations

Developers should be aware of the following limitations when using LVGL with FSP:

- Compilers may produce different results at this time (v9.3.0). AC6 is known to produce different software rendering output when blending certain image formats with a 32-bit color depth. LLVM is known to create artifacts with LVGL image decoding and blending.
- The DRW upstream driver does not support all LVGL rendering features. Some unsupported features are horizontal gradients, dashed lines, and blitting of certain image formats. In some cases, the LVGL software renderer will act as a fallback renderer, such as when an unsupported image format is used.

---

## Examples

### Basic Example

This is a basic example demonstrating the initialization of LVGL and the port layer within a FreeRTOS thread entry function, then servicing LVGL timer logic in a loop.

*Note*

*The port layer manages the GLCDC submodule internally, while the LVGL project manages the DRW submodule; they do not need to be opened directly.*

```
void lvgl_thread_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 fsp_err_t error;
 /* Initialize LVGL. */
    lv_init();
 /* TODO: Perform any initialization necessary for the physical display. */
 /* Initialize the LVGL port driver. */
    error = RM_LVGL_PORT_Open(&g_lvgl_port_ctrl, &g_lvgl_port_cfg);
```

```
 /* TODO: Handle fsp_err_t return values as required. */
    (void) error;
 /* TODO: Create an input device using LVGL. */
 /* TODO: Create a user defined UI using LVGL widgets. */
 /* For example, add a label to the LVGL active screen on the default display. */
    lv_obj_t * active_screen = lv_screen_active();
    lv_obj_set_style_text_color(active_screen, lv_color_black(), 0);
    lv_obj_set_style_bg_color(active_screen, lv_color_white(), 0);
    lv_obj_t * label_object = lv_label_create(active_screen);
    lv_label_set_text(label_object, "Hello World!");
 /* Call lv_timer_handler() to process LVGL timer tasks, which includes display
refresh. */
 while (1)
    {
       lv_timer_handler();
       vTaskDelay(1);
    }
}
```

*Note*

*For further example code please consult the LVGL Project Documentation.*

### Data Structures

| | |
|---|---|
| struct | rm_lvgl_port_callback_args_t |
| struct | rm_lvgl_port_cfg_t |
| struct | rm_lvgl_port_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_lvgl_port_device_t |
| enum | rm_lvgl_port_event_t |

## Data Structure Documentation

### ◆ rm_lvgl_port_callback_args_t

struct rm_lvgl_port_callback_args_t

| Callback arguments for the FSP LVGL Port | | |
|---|---|---|
| Data Fields | | |
| rm_lvgl_port_device_t | device | Device code. |
| rm_lvgl_port_event_t | event | Event code of the low level hardware. |
| uint32_t | error | Error code if RM_LVGL_PORT_EVENT_ERROR. |

### ◆ rm_lvgl_port_cfg_t

| struct rm_lvgl_port_cfg_t |
|---|
| Configuration structure for the FSP LVGL Port |

### ◆ rm_lvgl_port_instance_ctrl_t

| struct rm_lvgl_port_instance_ctrl_t |
|---|
| Instance control block for LVGL Port |

## Enumeration Type Documentation

### ◆ rm_lvgl_port_device_t

| enum rm_lvgl_port_device_t | |
|---|---|
| Low level device code for the LVGL Port | |
| Enumerator | |
| RM_LVGL_PORT_DEVICE_NONE | Non hardware. |
| RM_LVGL_PORT_DEVICE_DISPLAY | Display device. |

### ◆ rm_lvgl_port_event_t

| enum rm_lvgl_port_event_t | |
|---|---|
| Display event codes | |
| Enumerator | |
| RM_LVGL_PORT_EVENT_DISPLAY_VPOS | Display interface VPOS. |
| RM_LVGL_PORT_EVENT_UNDERFLOW | Display interface underflow. |

## Function Documentation

### ◆ RM_LVGL_PORT_Open()

fsp_err_t RM_LVGL_PORT_Open ( rm_lvgl_port_instance_ctrl_t *const *p_instance_ctrl*, rm_lvgl_port_cfg_t const *const *p_cfg* )

Open the LVGL port driver module.

Example:

```
/* Initialize the LVGL port driver. */
   error = RM_LVGL_PORT_Open(&g_lvgl_port_ctrl, &g_lvgl_port_cfg);
/* TODO: Handle fsp_err_t return values as required. */
   (void) error;
```

***Note***

*This function should be called after the LVGL function lv_init() has been called.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LVGL port successfully opened. |
| FSP_ERR_OUT_OF_MEMORY | Internal call to lv_display_create() failed to allocate memory for a display. |

See Common_Error_Codes for other possible return codes. This function calls R_GLCDC_Open(), R_GLCDC_Start(), and R_GLCDC_BufferChange().

## 5.2.8.8 MIPI Display Serial Interface (r_mipi_dsi)

Modules » Graphics

### Functions

| | |
|---|---|
| fsp_err_t | R_MIPI_DSI_Open (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_cfg_t const *const p_cfg) |
| fsp_err_t | R_MIPI_DSI_Close (mipi_dsi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MIPI_DSI_Start (mipi_dsi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MIPI_DSI_Stop (mipi_dsi_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MIPI_DSI_UlpsEnter (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_lane_t lane) |
| fsp_err_t | R_MIPI_DSI_UlpsExit (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_lane_t lane) |
| fsp_err_t | R_MIPI_DSI_Command (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_cmd_t *p_cmd) |

| | |
|---|---|
| fsp_err_t | R_MIPI_DSI_StatusGet (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_status_t *p_status) |

### Detailed Description

Driver for the MIPI DSI peripheral on RA MCUs. This module implements the Display Interface.

# Overview

The MIPI DSI peripheral consists of the Display Serial Interface (DSI-2) Host, physical layer (D-PHY), and supporting sub-systems. Together, these form a high-speed graphics serial bus that formats data from the GLCDC layer and sends it to an external display. The DSI-2 and D-Phy peripherals support MIPI Alliance for Display Serial Interface 2 Version 1.1 (with Errata 01) and MIPI Alliance Specification for D-PHY Version 2.1 (with Errata 01).

### Features

The following features are available:

| Feature | Options |
|---|---|
| Pixel formats | RGB888, RGB666, RGB565 |
| Number of lanes | Up to 2 |
| Maximum resolution | See GLCDC specifications |
| Maximum bandwidth | Up to 750 Mbps per high-speed lane |

Features:

- Supports up to 2 Lanes.
- Bidirectional LP mode transfer/receipt (LP-TX / LP-RX).
- Unidirectional High-Speed mode transfer (HS-TX).
- Data rates of up to 720 Mbps per lane.
- Ultra-Low-Power State (ULPS).
- Video input from GLCDC (RGB888, RGB666, RGB565).
- ECC/Checksum generation and verification for packets.
- Generation of scrambled packets.
- Automatic transition from HP to LP during blanking/porch periods.

# Configuration

### Build Time Configurations for r_mipi_dsi

The following build time configurations are defined in fsp_cfg/r_mipi_dsi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Graphics > MIPI Physical Layer (r_mipi_phy)

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_mipi_phy0 | Module name. |
| Timing | | | |
| Timing > THSPREP | | | |
| ns | Value must be a number, greater than or equal to zero. | 40 | (Nanosecond portion) Duration of the data lane LP-00 state, immediately before entry to the HS-0 state (ns) |
| UI | Value must be a number, greater than or equal to zero. | 5 | (UI portion) Duration of the data lane LP-00 state, immediately before entry to the HS-0 state (UI) |
| Timing > THSSETT | | | |
| ns | Value must be a number, greater than or equal to zero. | 200 | (Nanosecond portion) Period over which a transition to the HS state is to be ignored after the TCLK-PREPARE period has begun in the data lane. (ns) |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Period over which a transition to the HS state is to be ignored after the TCLK-PREPARE period has begun in the data lane. (UI) |
| Timing > THSZERO | | | |
| ns | Value must be a number, greater than or equal to zero. | 140 | (Nanosecond portion) Specify the data lane zero time before sending data (ns). |
| UI | Value must be a number, greater than or equal to zero. | 10 | (UI portion) Specify the data lane zero time before sending data (UI). |
| Timing > THSTRAIL | | | |
| ns | Value must be a | 60 | (Nanosecond portion) |

| | | | |
|---|---|---|---|
| | number, greater than or equal to zero. | | Specify the data lane trail time before exiting HS mode (ns). |
| UI | Value must be a number, greater than or equal to zero. | 4 | (UI portion) Specify the data lane trail time before exiting HS mode (UI). |
| Timing > TCLKPOST | | | |
| ns | Value must be a number, greater than or equal to zero. | 60 | (Nanosecond portion) Specify the duration after HS data lane trail time elapses before stopping the clock lane (ns). |
| UI | Value must be a number, greater than or equal to zero. | 52 | (UI portion) Specify the duration after HS data lane trail time elapses before stopping the clock lane (UI). |
| Timing > TCLKPRE | | | |
| ns | Value must be a number, greater than or equal to zero. | 0 | (Nanosecond portion) Specify the time clock is active before transitioning data lane into HS mode (ns). |
| UI | Value must be a number, greater than or equal to zero. | 8 | (UI portion) Specify the time clock is active before transitioning data lane into HS mode (UI). |
| Timing > TCLKPREP | | | |
| ns | Value must be a number, greater than or equal to zero. | 75 | (Nanosecond portion) Duration of the clock lane LP-00 state, immediately before entry to the HS-0 state (ns) |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Duration of the clock lane LP-00 state, immediately before entry to the HS-0 state (UI) |
| Timing > TCLKSETT | | | |
| ns | Value must be a number, greater than or equal to zero. | 500 | (Nanosecond portion) Duration over which a transition to the HS state is to be ignored |

| | | | |
|---|---|---|---|
| | | | after the TCLK_PREPARE period has begun in the clock lane (ns). |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Duration over which a transition to the HS state is to be ignored after the TCLK_PREPARE period has begun in the clock lane (ui). |
| Timing > TCLKMISS | | | |
| ns | Value must be a number, greater than or equal to zero. | 300 | (Nanosecond portion) Period from detection of the absence of a clock until disabling of the HS-RX in the clock lane (ns). |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Period from detection of the absence of a clock until disabling of the HS-RX in the clock lane (ui). |
| Timing > TLPX | | | |
| ns | Value must be a number, greater than or equal to zero. | 60 | (Nanosecond portion) Specify the time for the clock lane to exit low power mode (ns). |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Specify the time for the clock lane to exit low power mode (UI). |
| Timing > TCLKTRL | | | |
| ns | Value must be a number, greater than or equal to zero. | 60 | (Nanosecond portion) Specify the time after clock lane stop before exiting HS mode (ns). |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Specify the time after clock lane stop before exiting HS mode (UI). |
| Timing > TCLKZERO | | | |
| ns | Value must be a number, greater than or equal to zero. | 230 | (Nanosecond portion) Specify the time clock lane is zero before starting in HS mode (ns). |

| | | | |
|---|---|---|---|
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Specify the time clock lane is zero before starting in HS mode (UI) |
| Timing > THSEXIT | | | |
| ns | Value must be a number, greater than or equal to zero. | 100 | (Nanosecond portion) Specify the data lane HS mode exit time (ns). |
| UI | Value must be a number, greater than or equal to zero. | 0 | (UI portion) Specify the data lane HS mode exit time (UI) |
| LP Clock Divider | Value must be an integer. | 5 | Specify the MIPI PHY LP clock division ratio (Resulting frequency must be from 2-17 MHz) |
| TINIT (ns) | Value must be a number, greater than or equal to zero. | 600000 | Minimum duration of the TINIT state (ns) |
| DSI PLL Frequency (MHz) | Value must be between 160 MHz and 1440.0 MHz. | 1000.00 | Specify the MIPI PHY PLL frequency in MHz. |

**Configurations for Graphics > MIPI Display (r_mipi_dsi)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_mipi_dsi0 | Module name. |
| Options | | | |
| LCD External Clock Hz | Value must be an integer. | 0 | Specify the GLCDC external clock frequency in Hz (Set to 0 when GLCDC clock source is set to Internal). |
| Data Scramble Enable | • Disable<br>• Enable | Disable | Data Scramble Enable. Do not enable unless peripheral has data scramble function. |
| EoTP Enable | • Disable<br>• Enable | Enable | Disable to support devices that do not support EoTP transmission. |
| ECC Check Enable | • Disable<br>• Enable | Enable | ECC Check support enable. |

| | | | |
|---|---|---|---|
| CRC Enable | • Virtual Channel 0<br>• Virtual Channel 1<br>• Virtual Channel 2<br>• Virtual Channel 3 | | |
| Maximum Return Packet Size | Value must be an integer. | 1 | Specify the maximum return packet size to be received in LP-RX mode. |
| External Tearing Effect Detection Sense Select | • Rising Edge<br>• Falling Edge | 1 | Specify the maximum return packet size to be received in LP-RX mode. |
| HS-TX Timeout Count (us) | Value must be an integer. | 0 | Set LP-RX Timeout (LRX-H_TO) value. (0 is disabled) |
| LP-RX Host Processor Timeout (us) | Value must be an integer. | 0 | Set LP-RX Timeout (LRX-H_TO) value. (0 is disabled) |
| Turnaround Acknowledge Timeout (us) | Value must be an integer. | 0 | Set Turnaround Acknowledge Timeout value. (0 is disabled) |
| Peripheral Response Timeout (us) | Value must be an integer. | 0 | Set Peripheral Response Timeout BTA value. (0 is disabled) |
| LP Write Response Timeout (us) | Value must be an integer. | 0 | Set Low Power Write Acknowledge Timeout value. (0 is disabled) |
| LP Read Response Timeout (us) | Value must be an integer. | 0 | Set Low Power Read Acknowledge Timeout value. (0 is disabled) |
| HS Write Response Timeout (us) | Value must be an integer. | 0 | Set High Speed Write Acknowledge Timeout value. (0 is disabled) |
| HS Read Response Timeout (us) | Value must be an integer. | 0 | Set High Speed Read Acknowledge Timeout value. (0 is disabled) |
| Low Power | | | |
| Ultra Low Power State Wakeup Period (us) | Value must be a positive integer or zero. | 1000 | Set ultra low power state wakeup period (us). |
| Clock Lane | | | |
| Continuous Mode | • Disable | Enable | Enable or disable |

| | | | |
|---|---|---|---|
| | • Enable | | continuous clock mode. |
| Data Lane | | | |
| Number of Data Lanes | Value must be an integer. | 2 | Specify the number of data lanes. Note: not all data lanes are capable of HS operation. See Usage Notes for additional information. |
| Video Mode | | | |
| Video Mode > Pixel Packet | | | |
| Sync Pulse | • HSE and VSE are not transmitted<br>• HSE and VSE are transmitted | HSE and VSE are not transmitted | Select if HSE and VSE are transmitted. Disable for Burst Mode sequence or Non-Burst Mode with Sync Events. |
| Virtual Channel ID | Value must be an integer. | 0 | Select the video mode virtual channel ID. |
| Delay Override (0 to disable) | Value must be an integer. | 0 | Override FSP calculated delay value (not recommended for most users). Delay for DSI Host between first data reception from display module until DSI output begins. (Unit: 32xUI). Set to 0 to use FSP calculated value (recommended). |
| Prevent LP Transition | • No LP during the HSA period<br>• No LP during the HBP period<br>• No LP during the HFP period | | Prevent LP transition during specified periods. |
| Interrupts | | | |
| dsi_seq0 Interrupt Priority | MCU Specific Options | | Select the Low-Power Sequence command operation interrupt priority. |
| dsi_seq1 Interrupt Priority | MCU Specific Options | | Select the High-Speed Sequence command operation interrupt priority. |
| dsi_vin1 Interrupt Priority | MCU Specific Options | | Select the Video Input interrupt priority. |
| dsi_rcv Interrupt | MCU Specific Options | | Select the Receive |

| | | | |
|---|---|---|---|
| Priority | | | interrupt priority. |
| dsi_ferr Interrupt Priority | MCU Specific Options | | Select the Fatal Error interrupt priority. |
| dsi_ppi Interrupt Priority | MCU Specific Options | | Select the PHY-Protocol Interface interrupt priority. |
| Receive Interrupt Enable | Refer to the RA Configuration tool for available options. | Multiple options are selected; refer to the RA Configuration tool for selected options. | Enable receive interrupts. |
| Fatal Error Interrupt Enable | • HS TX Timeout<br>• LP-RX Host Processor Timeout<br>• Turnaround Acknowledge Timeout<br>• Escape mode Entry Error<br>• LPDT Sync Error<br>• Control Error<br>• LP0 Contention Error<br>• LP1 Contention Error | • HS TX Timeout<br>• LP-RX Host Processor Timeout<br>• Turnaround Acknowledge Timeout<br>• Escape mode Entry Error<br>• LPDT Sync Error<br>• Control Error<br>• LP0 Contention Error<br>• LP1 Contention Error | Enable Fatal Error interrupts. |
| Physical Lane Interrupt Enable | • Data Lane-0 Rx to Tx Transition<br>• Data Lane-0 Tx to Rx Transition<br>• Clock Lane ULPS Enter<br>• Clock Lane ULPS Exit<br>• Clock Lane LP to HS Transition<br>• Clock Lane HS to LP Transition<br>• Data Lane ULPS Enter<br>• Data Lane ULPS Exit | | Enable Physical Lane interrupts. |
| Video Mode Interrupt Enable | • Video Mode Operation Start<br>• Video Mode Operation Stop<br>• Video Mode Operation Ready<br>• Timing Error<br>• Video Buffer Underflow Error | • Video Buffer Underflow Error<br>• Video Buffer Overflow Error | Enable Video Mode interrupts. |

| | | | |
|---|---|---|---|
| | • Video Buffer Overflow Error | | |
| Sequence Channel 0 Interrupt Enable | • All Actions Finish<br>• All Descriptors Finish<br>• Tx Internal Bus Error<br>• Receive Fatal Error<br>• Receive Fail<br>• Receive Packet Data Fail<br>• Receive Correctable Error Interrupt<br>• Receive Acknowledge and Error Report Packet | • All Actions Finish<br>• All Descriptors Finish<br>• Tx Internal Bus Error<br>• Receive Fatal Error<br>• Receive Fail<br>• Receive Packet Data Fail<br>• Receive Correctable Error Interrupt<br>• Receive Acknowledge and Error Report Packet | Enable Sequence Channel 0 interrupts. |
| Sequence Channel 1 Interrupt Enable | • All Actions Finish<br>• All Descriptors Finish<br>• Packet Size Error<br>• Tx Internal Bus Error<br>• Receive Fatal Error<br>• Receive Fail<br>• Receive Packet Data Fail<br>• Receive Correctable Error Interrupt<br>• Receive Acknowledge and Error Report Packet | • All Actions Finish<br>• All Descriptors Finish<br>• Packet Size Error<br>• Tx Internal Bus Error<br>• Receive Fatal Error<br>• Receive Fail<br>• Receive Packet Data Fail<br>• Receive Correctable Error Interrupt<br>• Receive Acknowledge and Error Report Packet | Enable Sequence Channel 1 interrupts. |
| Callback | Name must be a valid C symbol | mipi_dsi0_callback | A user callback function. If this callback function is provided it is called from the interrupt service routine (ISR) each time any interrupt occurs. |
| Callback Context | Name must be a valid C symbol | NULL | Pointer to the context structure to be passed through the callback argument. |

### Clock Configuration

The MIPI DSI D-PHY has a dedicated regulator (D-PHY LDO) and PLL (D-PHY PLL), which are managed by the driver. The D-PHY PLL frequency must be configured between 160 MHz and 1.44 GHz.

*Note*

*The D-PHY High-Speed data transmission rate is determined by the following formula: Line rate [Mbps] = fDPHYPLL [MHz] / 2*

### Pin Configuration

Communication to the external display occurs via one or more data lanes and one clock lane. Each of these lanes has dedicated pins. Lane 0 is capable of low-power data transfer and bidirectional communication with a display. Lane 1 is capable of low-power or high-speed data transfer to the external display. Additionally, an optional tearing effect connection (DSI_TE) may be used with this module.

## Usage Notes

### Display Data

The DSI-2 Host consumes data from the GLCDC module and prepares it for output via the D-PHY and connections to the display.

### MIPI DSI Operating Modes

MIPI DSI is capable of several operating modes: Non-Burst Mode with Sync Pulse, Non-Burst Mode with Sync Event, and Burst Mode. Each operational mode is achieved by configuring the peripheral with specific timimg and option settings.

Non-Burst Mode with Sync Pulse:

- GLCDC Video Clock bandwidth == MIPI Phy PLL bandwidth
- Sync Pulse is enabled (HSE and VSE are transmitted)

Non-Burst Mode with Sync Event:

- GLCDC Video Clock bandwidth == MIPI Phy PLL bandwidth
- Sync Pulse is disabled (HSE and VSE are not transmitted)

Burst Mode:

- GLCDC Video Clock bandwidth < MIPI Phy PLL bandwidth
- Sync Pulse is disabled (HSE and VSE are not transmitted)

For the purpose of this section:

- GLCDC Video Clock bandwidth is defined as: (Panel Clock MHz) * (Bits per pixel)
- MIPI Phy PLL bandwidth is defined as: (MIPI Phy PLL Clock MHz / 2) * (Number of MIPI data lanes) * 8 - (Configuration Dependent Transmission Overhead)

*Note*

*Consult the MIPI DSI Specification and the connected display's User Manual to calculate the desired bandwidth utilization. Additionally, especially with strict timing requirements, it is recommended to empirically validate the configuration and use the API to check for reported errors.*

### MIPI PHY Data Lanes

The DSI-2 Host supports two basic types of operations: Command Mode and Video Mode. While a data lane is in Low-Power (LP) operation, Command Mode may be used for bi-directional communication with a connected display using a pre-defined set of command descriptors.

*Note*

*GLCDC Video Clock bandwidth must not exceed Data Bus bandwidth or MIPI Phy PLL bandwidth.*

### MIPI PHY Timing Configuration

The MIPI DSI D-PHY configuration controls timing aspects of Low Power (LP) and High Speed (HS) communication. Configure these values to match specifications listed in the datasheet for the display.

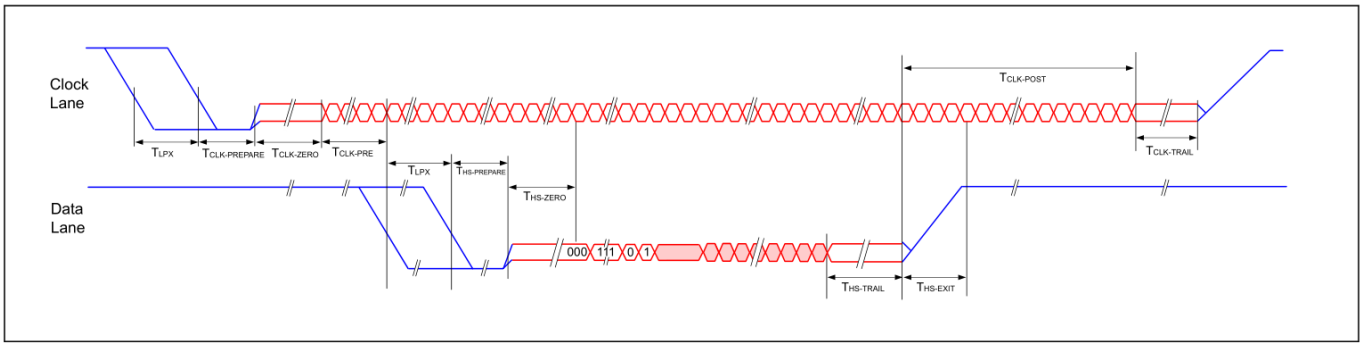



Figure 258: High-Speed data transmission in normal mode

### Command Mode Operation

Two internal channels may be used for command mode operations, available for all physical lane configurations. Channel-0 supports only LP mode (LP-TX, LP-RX), while Channel-1 supports LP mode (LP-TX, LP-RX) and HS mode (HS-TX).

There are two basic packet formats, short and long. Each format may be transmitted in high-speed or low-power modes. Packets may be followed by a Bus Turn-Around (BTA) request for reading information from the display. Once configured and started, video packets are transmitted automatically, until video output is stopped.

In addition to the full set of MIPI DSI commands, the application may trigger any of four special commands by setting flags in the message structure. These special commands are Reset Signal, Initial Skew Calibration, Periodic Skew Calibration, and No-Operation.

*Note*

*For peripherals with more than one lane, physical Lane 0 is used for all peripheral-to-processor transmissions. Other lanes are unidirectional, from the host processor to the peripheral.*

### Acknowledge and Error Reporting

The application is notified of Acknowledge and Error Reporting (AwER) via an optional receive interrupt event. The most recent and accumulated AwER data may be retrieved by calling R_MIPI_DSI_StatusGet(). The application may send a MIPI_DSI_CMD_FLAG_BTA_NO_WRITE message with tx size of zero to request AwER from the peripheral.

### Ultra-low Power State

Ultra-low Power State (ULPS) may be actived when HS and LP operations are not occurring. Clock and Data lanes may be transitioned into ULPS independently from each other.

### Limitations

Developers should be aware of the following limitations when using the MIPI DSI API:

- MIPI DSI must be closed before transitioning MCU into low-power modes.
- Data bus bandwidth limitations should be considered when setting GLCDC and MIPI DSI clock speeds.
- When the MIPI DSI is used, voltage of VCC and AVCC_MIPI pin must be same and 3V or larger(VCC=AVCC_MIPI≧3V)

### Interrupt Configuration

When enabled, Interrupts will invoke the configured callback function. Low-power and High-Speed command status should be determined by checking Sequence 0 and Sequence 1 events, respectively. See the DSI Error Handling section of the user manual for information about how to handle error events.

### MIPI DSI Setup with External Display

Especially for use with display middleware such as emWin or GUIX, the callback will be invoked with post-open and pre-video-start events. Depending on your hardware, it may be necessary to use these events to configure the display.

## Examples

### Basic Example

This is a basic example showing the minimum code required to initialize and start the MIPI DSI module.

```
void mipi_dsi_minimal_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = R_MIPI_DSI_Open(&g_mipi_dsi0_ctrl, &g_mipi_dsi0_cfg);
    assert(FSP_SUCCESS == err);
 /* Application to perform display specific initialization. */
    uint8_t       cmd_tx_buffer[] = {0x30, 0x01}; // NOLINT(readability-magic-
numbers)
 mipi_dsi_cmd_t cmd             =
    {
       .channel    = 0,
       .cmd_id      = MIPI_CMD_ID_DCS_SHORT_WRITE_1_PARAM,
```

```
      .flags       = (mipi_dsi_cmd_flag_t) (MIPI_DSI_CMD_FLAG_LOW_POWER |
MIPI_DSI_CMD_FLAG_BTA_READ),
      .tx_len      = 1,
      .p_tx_buffer = cmd_tx_buffer,
   };
   err = R_MIPI_DSI_Command(&g_mipi_dsi0_ctrl, &cmd);
   assert(FSP_SUCCESS == err);
/* Wait for tx/rx complete before sending additional messages */
while (!message_tx_complete)
   {
      ;
   }
while (!message_rx_complete)
   {
      ;
   }
   err = R_MIPI_DSI_Start(&g_mipi_dsi0_ctrl);
   assert(FSP_SUCCESS == err);
/* Trigger status message from peripheral by sending LP BTA
 * NOTE: This is required for ack_err status data to be populated */
mipi_dsi_cmd_t read_cmd = {0};
   read_cmd.cmd_id = MIPI_CMD_ID_DCS_READ;
   read_cmd.flags |= MIPI_DSI_CMD_FLAG_BTA_NO_WRITE | MIPI_DSI_CMD_FLAG_LOW_POWER;
   message_rx_complete = false;
   err                 = R_MIPI_DSI_Command(&g_mipi_dsi0_ctrl, &read_cmd);
   assert(FSP_SUCCESS == err);
while (!message_rx_complete)
   {
      ;
   }
/* Read peripheral and local MIPI DSI status
 * Note: peripheral ack_err status is cleared each time it is read using StatusGet
*/
mipi_dsi_status_t status;
```

```
    err = R_MIPI_DSI_StatusGet(&g_mipi_dsi0_ctrl, &status);
    assert(FSP_SUCCESS == err);
    err = R_MIPI_DSI_Stop(&g_mipi_dsi0_ctrl);
    assert(FSP_SUCCESS == err);
    err = R_MIPI_DSI_UlpsEnter(&g_mipi_dsi0_ctrl, (mipi_dsi_lane_t)
(MIPI_DSI_LANE_CLOCK | MIPI_DSI_LANE_DATA_ALL));
    assert(FSP_SUCCESS == err);
    err = R_MIPI_DSI_UlpsExit(&g_mipi_dsi0_ctrl, (mipi_dsi_lane_t)
(MIPI_DSI_LANE_CLOCK | MIPI_DSI_LANE_DATA_ALL));
    assert(FSP_SUCCESS == err);
    err = R_MIPI_DSI_Close(&g_mipi_dsi0_ctrl);
    assert(FSP_SUCCESS == err);
}
void mipi_dsi0_callback (mipi_dsi_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case MIPI_DSI_EVENT_POST_OPEN:
     {
 /* Application to configure peripheral using necessary interface and commands */
           configure_dsi_peripheral();
 break;
     }
 case MIPI_DSI_EVENT_RECEIVE:
     {
 /* Application to perform receive processing */
           message_rx_complete = true;
 break;
     }
 case MIPI_DSI_EVENT_SEQUENCE_0:
     {
           message_tx_complete = (p_args->tx_status ==
MIPI_DSI_SEQUENCE_STATUS_DESCRIPTORS_FINISHED);
 break;
```

```
      }
 default:
      {
 break;
      }
    }
}
void configure_dsi_peripheral (void)
{
 /* Send necessary commands to configure LCD */
}
```

### Data Structures

| | |
|---|---|
| struct | mipi_dsi_irq_cfg_t |
| struct | mipi_dsi_extended_cfg_t |
| struct | mipi_dsi_instance_ctrl_t |

## Data Structure Documentation

### ◆ mipi_dsi_irq_cfg_t

| struct mipi_dsi_irq_cfg_t | | |
|---|---|---|
| MIPI DSI interrupt configuration | | |
| Data Fields | | |
| uint8_t | ipl | Interrupt priority. |
| IRQn_Type | irq | Interrupt vector number. |

### ◆ mipi_dsi_extended_cfg_t

| struct mipi_dsi_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for MIPI DSI. | | |
| Data Fields | | |
| mipi_dsi_irq_cfg_t | dsi_seq0 | Sequence 0 interrupt. |
| mipi_dsi_irq_cfg_t | dsi_seq1 | Sequence 1 interrupt. |
| mipi_dsi_irq_cfg_t | dsi_ferr | DSI Fatal Error interrupt. |
| mipi_dsi_irq_cfg_t | dsi_ppi | D-PHY PPI interrupt. |
| mipi_dsi_irq_cfg_t | dsi_rcv | Receive interrupt. |

| | | |
|---|---|---|
| mipi_dsi_irq_cfg_t | dsi_vin1 | Video Input Operation interrupt. |
| uint32_t | dsi_rxie | Receive interrupt enable configuration. |
| uint32_t | dsi_ferrie | Fatal error interrupt enable configuration. |
| uint32_t | dsi_plie | Physical lane interrupt enable configuration. |
| uint32_t | dsi_vmie | Video mode interrupt enable configuration. |
| uint32_t | dsi_sqch0ie | Sequence Channel 0 interrupt enable configuration. |
| uint32_t | dsi_sqch1ie | Sequence Channel 1 interrupt enable configuration. |

### ◆ mipi_dsi_instance_ctrl_t

| struct mipi_dsi_instance_ctrl_t | |
|---|---|
| MIPI DSI instance control block. | |
| **Data Fields** | |
| uint32_t | open |
| | Interface is open. |
| | |
| bool | data_ulps_active |
| | Data lane ULPS status. |
| | |
| bool | clock_ulps_active |
| | Data lane ULPS status. |
| | |
| mipi_dsi_lane_t | ulps_status |
| | Ultra-low Power State active status. |
| | |
| mipi_dsi_cfg_t const * | p_cfg |
| | Pointer to configuration structure used to open the interface. |
| | |
| void(* | p_callback )(mipi_dsi_callback_args_t *) |

| | |
|---|---|
| | Pointer to callback that is called when an adc_event_t occurs. |
| | |
| void * | p_context |
| | Pointer to context to be passed into callback function. |
| | |
| mipi_dsi_callback_args_t * | p_callback_memory |
| | Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory. |
| | |

### Function Documentation

#### ◆ R_MIPI_DSI_Open()

fsp_err_t R_MIPI_DSI_Open ( mipi_dsi_ctrl_t *const *p_api_ctrl*, mipi_dsi_cfg_t const *const *p_cfg* )

Initialize the MIPI DSI peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel was successfully opened. |
| FSP_ERR_ASSERTION | One or both of the parameters was NULL. |
| FSP_ERR_ALREADY_OPEN | The instance is already opened. |
| FSP_ERR_INVALID_STATE | Display module must be opened before DSI. |

#### ◆ R_MIPI_DSI_Close()

fsp_err_t R_MIPI_DSI_Close ( mipi_dsi_ctrl_t *const *p_api_ctrl*)

Close MIPI DSI and display data instances, disable interrupts, and power-off the module.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The channel is successfully closed. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_IN_USE | Operation in progress and must be stopped before closing. |

### ◆ R_MIPI_DSI_Start()

fsp_err_t R_MIPI_DSI_Start ( mipi_dsi_ctrl_t *const *p_api_ctrl*)

Start video output. Initialize Video Output Registers Perform sequence steps 3 to 5 from "Start of Video Mode Operation" in the MIPI DSI section of the relevant hardware manual.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_IN_USE | The physical interface is currently in use. |
| FSP_ERR_INVALID_STATE | DSI is already in video mode. |

### ◆ R_MIPI_DSI_Stop()

fsp_err_t R_MIPI_DSI_Stop ( mipi_dsi_ctrl_t *const *p_api_ctrl*)

Stop video output.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data is successfully written to the D/A Converter. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_IN_USE | DSI cannot be closed while ULPS is active. |

### ◆ R_MIPI_DSI_UlpsEnter()

fsp_err_t R_MIPI_DSI_UlpsEnter ( mipi_dsi_ctrl_t *const *p_api_ctrl*, mipi_dsi_lane_t *lane* )

Enter Ultra-low Power State (ULPS).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information read successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_INVALID_MODE | Invalid mode for transition. |

### ◆ R_MIPI_DSI_UlpsExit()

fsp_err_t R_MIPI_DSI_UlpsExit ( mipi_dsi_ctrl_t *const *p_api_ctrl*, mipi_dsi_lane_t *lane* )

Exit Ultra-low Power State (ULPS).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information read successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |

### ◆ R_MIPI_DSI_Command()

fsp_err_t R_MIPI_DSI_Command ( mipi_dsi_ctrl_t *const *p_api_ctrl*, mipi_dsi_cmd_t * *p_cmd* )

Send a command to the peripheral device.

***Note***

*p_data will be used as either write data or a read buffer depending on the data id.*
*p_data memory must not be updated until sequence operation is complete if byte_count is greater than 16.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Command(s) queued successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. cmd_id specifies a long packet but p_data is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |
| FSP_ERR_IN_USE | The physical interface is currently in use or video mode is in operation. |
| FSP_ERR_INVALID_POINTER | Invalid pointer provided |
| FSP_ERR_INVALID_ARGUMENT | Invalid message configuration |

### ◆ R_MIPI_DSI_StatusGet()

fsp_err_t R_MIPI_DSI_StatusGet ( mipi_dsi_ctrl_t *const *p_api_ctrl*, mipi_dsi_status_t * *p_status* )

Provide information about current MIPI DSI status.

Note: Accumulated Acknowledge and Error (AwER) Status is cleared by calling this function. Latest AwER status is only set upon reception from peripheral.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information read successfully. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance is not open. |

## 5.2.8.9 Parallel Data Capture (r_pdc)

Modules » Graphics

### Functions

| | |
|---|---|
| fsp_err_t | R_PDC_Open (capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg)<br>Powers on PDC, handles required initialization described in the hardware manual. More... |
| fsp_err_t | R_PDC_Close (capture_ctrl_t *const p_ctrl)<br>Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts. More... |
| fsp_err_t | R_PDC_CaptureStart (capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer)<br>Starts a capture. Enables interrupts. More... |
| fsp_err_t | R_PDC_StatusGet (capture_ctrl_t *const p_ctrl, capture_status_t *p_status) |
| fsp_err_t | R_PDC_CallbackSet (capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the PDC peripheral on RA MCUs. This module implements the CAPTURE Interface.

# Overview

The PDC peripheral supports interfacing with external cameras by accepting timing and data signals in order to capture incoming data. A callback is invoked every time a frame of data is accepted.

### Features

- Capture incoming data into a user defined buffer
- Data bytes per pixel can be configured
- Endianess of the incoming data can be specified
- Supports configuring capture width and height
- Supports configuring vertical and horizontal sync polarity
- Horizontal and Vertical position for image/data capture can be specified
- External clock to the camera module can be adjusted
- Choice between DMA and DTC to transfer out the captured data
- The specified user callback is invoked when a data frame is captured

# Configuration

### Build Time Configurations for r_pdc

The following build time configurations are defined in fsp_cfg/r_pdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Graphics > Parallel Data Capture (r_pdc)

This module can be added to the Stacks tab via New Stack > Graphics > Parallel Data Capture (r_pdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_pdc0 | Module name. |
| Input | | | |
| Input > Signal polarity | | | |
| HSYNC | • High<br>• Low | High | Specify the active polarity of the HSYNC signal. |
| VSYNC | • High<br>• Low | High | Specify the active polarity of the VSYNC signal. |
| Input > Capture Specifications | | | |

| | | | |
|---|---|---|---|
| Number of pixels to capture horizontally | Value must be an integer greater than 0 | 640 | Specify the number of horizontal pixels to capture. |
| Number of lines to capture vertically | Value must be an integer greater than 0 | 480 | Specify the number of vertical pixels to capture. |
| Horizontal pixel to start capture from | Value must be an integer | 0 | Specify the horizontal pixel to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured. |
| Line to start capture from | Value must be an integer | 0 | Specify the vertical line to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured. |
| Bytes per pixel | Value must be an integer greater than 0 | 2 | Specify the number of bytes per pixel of the captured image data. |
| Clock divider | • CLK/2<br>• CLK/4<br>• CLK/6<br>• CLK/8<br>• CLK/10<br>• CLK/12<br>• CLK/14<br>• CLK/16 | CLK/2 | Specify the clock divider for clock frequency of the clock input PCKO to the PDC peripheral. |
| Endianess | • Little<br>• Big | Little | Specify the endianness of the captured image data. |
| Output | | | |
| Output > Buffer | | | |
| Number of image buffers | Value must be an integer greater than 0 | 1 | Specify the number of buffers to create. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | g_pdc_user_callback | A user callback function must be provided. This callback is invoked for every successful frame capture and any error conditions |
| PDC Interrupt Priority | MCU Specific Options | | Select the PDC |

| | | |
|---|---|---|
| | | interrupt priority. |
| DTC Interrupt Priority | MCU Specific Options | Select the DTC interrupt priority. |

### Clock Configuration

The PDC peripheral module uses the PCLKB as its clock source. The maximum clock to the camera module is PCLKB / 2.

### Pin Configuration

The PCKO pin is a clock output and should be connected to the clock input of the camera. The PIXCLK pin is a clock input and should be connected to the output pixel clock of the camera. Likewise, the HSYNC and VSYNC pins must be connected to the horizontal and vertical sync signals of the camera, respectively. The PIXD0-PIXD7 pins are the 8-bit data bus input and should be connected to the relevant output pins of the camera.

*Note*

*Camera control and serial communication pins must be configured separately and are not controlled by this module.*

## Usage Notes

### Interrupt Configuration

- PDC error interrupts are used by this module for reporting errors such as overrun, underrun, vertical line number setting and horizontal byte number setting errors.
- In addition to the PDC error interrupts, DMA or DTC interrupts are also used internally to perform data transfer from this peripheral to the specified image buffer.
- Receive data ready interrupt is used as activation source for DMA and DTC trigger.

### Enabling Transfer Modules

- An option to select between DMAC or DTC is provided with DMA as the default transfer choice.
- For further details on DMA please refer Transfer (r_dmac)
- For further details on DTC please refer Transfer (r_dtc)

### PDC setup with external camera

- Before configuring the external camera device the PDC Open API must be called in order to start clock output.
- Ensure that the memory pointed to by p_buffer is both valid and large enough to store a complete image.
- The amount of space required (in bytes) can be calculated as: size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel
- Ensure that the size above is divisible by and aligned to 32 bytes.

## Examples

### Basic Example

This is a basic example of minimal use of the PDC in an application. This example shows how this

driver can be used for capturing data from an external I/O device such as an image sensor.

```
void g_pdc_user_callback (capture_callback_args_t * p_args)
{
 if (PDC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_capture_ready = true;
    }
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the PDC module */
    err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the camera module at this point. This implementation is camera vendor
specific. */
    camera_module_initialization();
 /* Initialize capture ready flag to false. This gets set to true in PDC callback
upon successful frame capture. */
    g_capture_ready = false;
    err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);
    assert(FSP_SUCCESS == err);
    uint32_t timeout_ms = PDC_DELAY_MS;
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_capture_ready) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (0U == timeout_ms)
    {
       __BKPT(0);
    }
```

```
}
static void camera_module_initialization (void)
{
 /* Camera vendor specific initialization to be done here */
}
```

### Data Structures

| | |
|---|---|
| struct | pdc_extended_cfg_t |
| struct | pdc_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | pdc_event_t |
| enum | pdc_clock_division_t |
| enum | pdc_endian_t |
| enum | pdc_hsync_polarity_t |
| enum | pdc_vsync_polarity_t |

## Data Structure Documentation

### ◆ pdc_extended_cfg_t

| struct pdc_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for PDC. | | |
| Data Fields | | |
| pdc_clock_division_t | clock_division | Clock divider. |
| pdc_endian_t | endian | Endian of capture data. |
| pdc_hsync_polarity_t | hsync_polarity | Polarity of HSYNC input. |
| pdc_vsync_polarity_t | vsync_polarity | Polarity of VSYNC input. |
| uint8_t | pdc_ipl | PDC interrupt priority. |
| uint8_t | transfer_req_ipl | Transfer interrupt priority. |
| IRQn_Type | pdc_irq | PDC IRQ number. |
| IRQn_Type | transfer_req_irq | Transfer request IRQ number. |
| transfer_instance_t const * | p_lower_lvl_transfer | Pointer to the transfer instance the PDC should use. |

### ◆ pdc_instance_ctrl_t

| struct pdc_instance_ctrl_t |
| --- |
| PDC instance control block. DO NOT INITIALIZE. |

## Enumeration Type Documentation

### ◆ pdc_event_t

| enum pdc_event_t | |
| --- | --- |
| PDC events | |
| Enumerator | |
| PDC_EVENT_TRANSFER_COMPLETE | Complete frame transferred by DMAC/DTC. |
| PDC_EVENT_RX_DATA_READY | Receive data ready interrupt. |
| PDC_EVENT_FRAME_END | Frame end interrupt. |
| PDC_EVENT_ERR_OVERRUN | Overrun interrupt. |
| PDC_EVENT_ERR_UNDERRUN | Underrun interrupt. |
| PDC_EVENT_ERR_V_SET | Vertical line setting error interrupt. |
| PDC_EVENT_ERR_H_SET | Horizontal byte number setting error interrupt. |

### ◆ pdc_clock_division_t

| enum pdc_clock_division_t | |
|---|---|
| Clock divider applied to PDC clock to provide PCKO output frequency | |
| Enumerator | |
| PDC_CLOCK_DIVISION_2 | CLK / 2. |
| PDC_CLOCK_DIVISION_4 | CLK / 4. |
| PDC_CLOCK_DIVISION_6 | CLK / 6. |
| PDC_CLOCK_DIVISION_8 | CLK / 8. |
| PDC_CLOCK_DIVISION_10 | CLK / 10. |
| PDC_CLOCK_DIVISION_12 | CLK / 12. |
| PDC_CLOCK_DIVISION_14 | CLK / 14. |
| PDC_CLOCK_DIVISION_16 | CLK / 16. |

### ◆ pdc_endian_t

| enum pdc_endian_t | |
|---|---|
| Endian of captured data | |
| Enumerator | |
| PDC_ENDIAN_LITTLE | Data is in little endian format. |
| PDC_ENDIAN_BIG | Data is in big endian format. |

### ◆ pdc_hsync_polarity_t

| enum pdc_hsync_polarity_t | |
|---|---|
| Polarity of input HSYNC signal | |
| Enumerator | |
| PDC_HSYNC_POLARITY_HIGH | HSYNC signal is active high. |
| PDC_HSYNC_POLARITY_LOW | HSYNC signal is active low. |

### ◆ pdc_vsync_polarity_t

| enum pdc_vsync_polarity_t | |
|---|---|
| Polarity of input VSYNC signal | |
| Enumerator | |
| PDC_VSYNC_POLARITY_HIGH | VSYNC signal is active high. |
| PDC_VSYNC_POLARITY_LOW | VSYNC signal is active low. |

## Function Documentation

### ◆ R_PDC_Open()

fsp_err_t R_PDC_Open ( capture_ctrl_t *const *p_ctrl*, capture_cfg_t const *const *p_cfg* )

Powers on PDC, handles required initialization described in the hardware manual.

Implements capture_api_t::open.

The Open function provides initial configuration for the PDC module. It powers on the module and enables the PCLKO output and the PIXCLK input. Further initialization requires the PIXCLK input to be running in order to be able to reset the PDC as part of its initialization. This clock is input from a camera module and so the reset and further initialization is performed in capture_api_t::captureStart. This function should be called once prior to calling any other PDC API functions. After the PDC is opened the Open function should not be called again without first calling the Close function.

Example:

```
/* Initialize the PDC module */
   err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more of the following parameters is NULL<br>1. p_cfg is NULL<br>2. p_api_ctrl is NULL<br>3. The pointer to the transfer interface in the p_cfg parameter is NULL<br>4. Callback parameter is NULL.<br>5. Invalid IRQ number assigned |
| FSP_ERR_INVALID_ARGUMENT | One or more of the following parameters is incorrect<br>1. bytes_per_pixel is zero<br>2. x_capture_pixels is zero<br>3. y_capture_pixels is zero<br>4. x_capture_start_pixel + x_capture_pixels is greater than 4095, OR<br>5. y_capture_start_pixel + y_capture_pixels is greater than 4095 |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_PDC_Close()

fsp_err_t R_PDC_Close ( capture_ctrl_t *const *p_ctrl*)

Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts.

Implements capture_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_api_ctrl is NULL |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |

### ◆ R_PDC_CaptureStart()

fsp_err_t R_PDC_CaptureStart ( capture_ctrl_t *const *p_ctrl*, uint8_t *const *p_buffer* )

Starts a capture. Enables interrupts.

Implements capture_api_t::captureStart.

Sets up the transfer interface to transfer data from the PDC into the specified buffer. Configures the PDC settings as previously set by the capture_api_t::open API. These settings are configured here as the PIXCLK input must be active for the PDC reset operation. When a capture is complete the callback registered during capture_api_t::open API call will be called.

Example:

```
err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Capture start successful. |
| FSP_ERR_ASSERTION | One or more of the following parameters is NULL<br>1. p_api_ctrl is NULL<br>2. p_buffer is NULL |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |
| FSP_ERR_IN_USE | PDC transfer is already in progress. |
| FSP_ERR_TIMEOUT | Reset operation timed out. |
| FSP_ERR_NOT_INITIALIZED | Callback function has not been set |

### ◆ R_PDC_StatusGet()

fsp_err_t R_PDC_StatusGet ( capture_ctrl_t *const *p_ctrl*, capture_status_t * *p_status* )

Provides the pdc operating status.

Implements capture_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One or more parameters is NULL. |
| FSP_ERR_NOT_OPEN | Open has not been successfully called. |

### ◆ R_PDC_CallbackSet()

fsp_err_t R_PDC_CallbackSet ( capture_ctrl_t *const *p_ctrl*, void(*)(capture_callback_args_t *) *p_callback*, void *const *p_context*, capture_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure.

Implements capture_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This is just a stub at present |

#### 5.2.8.10 SEGGER emWin RA Port (rm_emwin_port)

Modules » Graphics

SEGGER emWin port for RA MCUs.

# Overview

The SEGGER emWin RA Port module provides the configuration and hardware acceleration support necessary for use of emWin on RA products. The port provides full integration with the graphics peripherals as well as FreeRTOS.

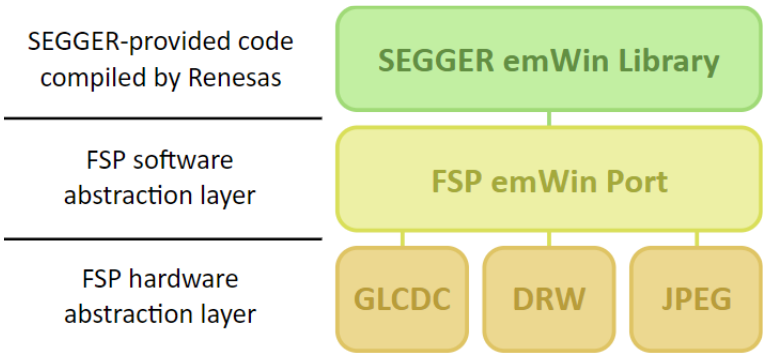



Figure 259: SEGGER emWin FSP Port Block Diagram (using RA6M3 as example)

*Note*

*This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. Please consult the SEGGER emWin User Guide (UM03001) for details on how to use emWin in your project.*

### Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- DRW Engine (r_drw)
  - Drawing bitmaps (ARGB8888 and RGB565)
  - 4bpp font rendering
  - Rectangle fill
  - Line and shape drawing
  - Anti-aliased operations
    - Circle stroke and fill
    - Polygon stroke and fill
    - Lines and arcs
- JPEG Codec (r_jpeg)
  - JPEG decoding
- Graphics LCD Controller (r_glcdc)
  - Brightness, contrast and gamma correction
  - Pixel format conversion (framebuffer to LCD)

## Configuration

### Build Time Configurations for rm_emwin_port

The following build time configurations are defined in fsp_cfg/rm_emwin_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Memory Allocation | | | |
| GUI Heap Size | Value must be a non-negative integer | 32768 | Set the size of the heap to be allocated for use exclusively by emWin. |
| Section for GUI Heap | Manual Entry | .ram_noinit | Specify the section in which to allocate the GUI heap. |

| | | | |
|---|---|---|---|
| Maximum Layers | Value must be a non-negative integer | 16 | Set the maximum number of available display layers in emWin.<br><br>This setting is not related to GLCDC Layer 1 or 2. |
| AA Font Conversion Buffer Size | Value must be a non-negative integer | 400 | Set the size of the conversion buffer for anti-aliased font glyphs. This should be set to the size (in bytes) of the largest AA character to be rendered. |
| LCD Settings | | | |
| Wait for Vertical Sync | • Enabled<br>• Disabled | Enabled | When enabled emWin will wait for a vertical sync event each time the display is updated. If an RTOS is used the thread will yield; otherwise each frame will block until Vsync.<br><br>WARNING: Disabling vertical sync will result in tearing. It is recommended to always leave this setting Enabled if an RTOS is used. |
| JPEG Decoding | | | |
| JPEG Decoding > General | | | |
| Input Alignment | • 8-byte aligned (faster)<br>• Unaligned (slower) | 8-byte aligned (faster) | Setting this option to 8-bit alignment can allow the hardware JPEG Codec to directly read JPEG data. This speeds up JPEG decoding operations and reduces RAM overhead, but all JPEG images must reside on an 8-byte boundary.<br><br>When this option is enabled the input buffer is not allocated. |

| | | | |
|---|---|---|---|
| Double-Buffer Output | • Enabled<br>• Disabled | Disabled | Enable this option to configure JPEG decoding operations to use a double-buffered output pipeline. This allows the JPEG to be rendered to the display at the same time as decoding at the cost of additional RAM usage.<br><br>Enabling this option automatically allocates double the output buffer size. |
| Error Timeout | Value must be a non-negative integer | 50 | Set the timeout for JPEG decoding operations (in RTOS ticks) in the event of a decode error. |
| JPEG Decoding > Buffers | | | |
| Input Buffer Size | Value must be a non-negative integer | 0x1000 | Set the size of the JPEG decode input buffer (in bytes). This buffer is used to ensure 8-byte alignment of input data. Specifying a size smaller than the size of the JPEG to decode will use additional interrupts to stream data in during the decoding process. |
| Output Buffer Size | Value must be a non-negative integer | 0x3C00 | Set the size of the JPEG decode output buffer (in bytes). An output buffer smaller than the size of a decoded image will use additional interrupts to stream the data into a framebuffer.<br><br>Unless you are sure of the subsampling used in and the size of the input JPEG images it is recommended to allocate at least 16 framebuffer lines of memory. |
| Section for Buffers | Manual Entry | .ram_noinit | Specify the section in |

which to allocate the JPEG work buffers.

### Hardware Configuration

No clocks or pins are directly required by this module. Please consult the submodules' documentation for their requirements.

### Library Configuration

emWin is provided as a pre-compiled library. To maximize compatibility the build-time options are configured as follows:

```
#define GUI_OS (1) // Context switch support enabled
#define GUI_MAXTASK (1) // One task supported by default (can be increased at runtime
via GUITASK_SetMaxTask())
#define GUI_NUM_LAYERS (3) // Up to three displays supported
#define GUI_SUPPORT_TOUCH (1) // Support touch screens
#define GUI_SUPPORT_MOUSE (1) // Support a mouse
#define GUI_SUPPORT_MEMDEV (1) // Memory devices available
#define GUI_WINSUPPORT (1) // Window manager available
#define GUI_SUPPORT_BIDI (1) // Bidirectional text enabled
#define GUI_DEBUG_LEVEL (2) // Parameter and consistency checks enabled (no logging)
```

All other options are left at the default settings in GUI_ConfDefaults.h.

## Usage Notes

### Getting Started

To get started with emWin in an RA project the following must be performed:

1. Open the RA Configuration editor for your project
2. Add emWin to your project in the Stacks view by clicking **New Stack -> SEGGER -> SEGGER emWin**
3. Ensure the configuration options for emWin are set as necessary for your application
4. Set the properties for the GLCDC module to match the timing and memory requirements of your display panel
5. Set the JPEG decode color depth to the desired value (if applicable)
6. Ensure interrupts on all modules are enabled:
    - GLCDC Vertical Position (Vpos) Interrupt
    - DRW Interrupt (if applicable)
    - JPEG Encode/Decode and Data Transfer Interrupts (if applicable)
7. Confirm stack and heap are configured as needed
    - When starting development a minimum stack of 0x1000 (4K) and heap of 0x4000 (16K) are recommended until actual usage can be characterized
8. Click Generate Project Content to save and commit configuration changes

At this point the project is now ready to build with emWin. Please refer to the SEGGER emWin User Guide (UM03001) as well as demo and sample code for details on how to create a GUI application.

### Using Hardware Acceleration

In most cases there is no need to perform additional configuration to ensure the DRW Engine is used. However, there are some guidelines that should be followed depending on the item in question:

- Bitmaps:
  - ARGB8888, RGB888 and RGB565 bitmaps require no additional settings.
- Anti-aliased shapes:
  - Anti-aliased lines, circles, polygons, polygon outlines and arcs are rendered with the DRW Engine.
- Anti-aliased (4bpp) fonts:
  - Set the text mode to GUI_TM_TRANS or create the relevant widget with WM_CF_HASTRANS set.
  - Ensure the "AA Font Conversion Buffer Size" configuration option is set to a size equal to or greater than the size (in bytes) of the largest glyph.
- 8bpp palletized images:
  - When creating these images ensure transparency is not enabled as the SEGGER method for this is not compatible with the DRW Engine.
- RLE-encoded images:
  - Hardware acceleration is not available for SEGGER's RLE format.
- JPEG images:
  - Align any user-declared JPEG data to an 8-byte boundary. If 8-byte alignment cannot be guaranteed disable the **JPEG Decoding -> General -> Input Alignment** option in the RA Configuration.

### Multi-thread Support

When the "Multi-thread Support" configuration is enabled, emWin can be called from multiple threads. This comes with advantages and disadvantages:

Advantages:

- High flexibility in development of applications
- Threads can pend and post on emWin events

Disadvantages:

- Slightly higher RAM/ROM use
- Large GUI projects can become difficult to debug

*Note*

*Multi-thread support is independent of RTOS support. RTOS support is managed internally and cannot be manually configured.*

### Limitations

Developers should be aware of the following limitations when using SEGGER emWin with FSP:

- Hardware acceleration is not available when using color modes lower than 16 bits.
- Hardware acceleration is not available for SEGGER's RLE image format.
- Rotated screen modes are not supported.
- Because emWin is provided as a pre-compiled library the build-time options are fixed. See

the Library Configuration section for the supplied configuration.

# Examples

### Basic Example

This is a basic example demonstrating a very simple emWin application. The screen is cleared to white and "Hello World!" is printed in the center.

*Note*

*emWin manages the GLCDC, DRW and JPEG Codec submodules internally; they do not need to be opened directly.*

```
#include "DIALOG.h"
#define COLOR_WHITE 0x00FFFFFFU
#define COLOR_BLACK 0x00000000U
#define GUI_DRAW_DELAY 100
static void _cbMain (WM_MESSAGE * pMsg)
{
    GUI_RECT Rect;
 switch (pMsg->MsgId)
    {
 case WM_CREATE:
      {
 break;
       }
 case WM_PAINT:
      {
 /* Clear background to white */
            GUI_SetBkColor(COLOR_WHITE);
            GUI_Clear();
 /* Draw "Hello World!" in black in the center */
            WM_GetClientRect(&Rect);
            GUI_SetColor(COLOR_BLACK);
            GUI_DispStringInRect("Hello World!", &Rect, GUI_TA_VCENTER |
GUI_TA_HCENTER);
 break;
```

```
      }
 default:
     {
          WM_DefaultProc(pMsg);
 break;
      }
   }
}
void emWinTask (void)
{
   int32_t xSize;
   int32_t ySize;
 /* Initialize emWin */
   GUI_Init();
 /* Get screen dimensions */
   xSize = LCD_GetXSize();
   ySize = LCD_GetYSize();
 /* Create main window */
   WM_CreateWindowAsChild(0, 0, xSize, ySize, WM_HBKWIN, WM_CF_SHOW, _cbMain, 0);
 /* Enter main drawing loop */
 while (1)
   {
      GUI_Delay(GUI_DRAW_DELAY);
   }
}
```

*Note*

*For further example code please consult SEGGER emWin documentation, which can be downloaded here, as well as the Quick Start Guide and example project(s) provided with your Evaluation Kit (if applicable).*

### 5.2.8.11 Segment LCD (r_slcdc)

Modules » Graphics

#### Functions

| | |
|---|---|
| fsp_err_t | R_SLCDC_Open (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SLCDC_Write (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count) |
| fsp_err_t | R_SLCDC_Modify (slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data, uint8_t const data_mask) |
| fsp_err_t | R_SLCDC_Start (slcdc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SLCDC_Stop (slcdc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SLCDC_SetContrast (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast) |
| fsp_err_t | R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area) |
| fsp_err_t | R_SLCDC_Close (slcdc_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface.

# Overview

The segment LCD controller (SLCDC) utilizes two to four reference voltages to provide AC signals for driving traditional segment LCD panels. Depending on the LCD and MCU package, up to 272 segments can be driven. A built-in link to the RTC allows for up to 152 segments to switch between two patterns at regular intervals. An on-chip boost driver can be used to provide configurable reference voltages up to 5.25V allowing for simple contrast adjustment.

### Features

The SLCDC module can perform the following functions:

- Initialize, start and stop the SLCDC
- Set and modify the output pattern
- Blink between two patterns based on a periodic RTC interrupt signal
- Adjust display contrast (only when using internal voltage boosting)
- Select reference voltage mode: VL1 reference mode (1/3 or 1/4 Bias) and VL2 reference mode (1/3 Bias) can be selected at internal voltage boosting method, conventional VCC reference mode (1/3 Bias) and VL4 reference mode (1/3) Bias can be selected at capacitor split method

# Configuration

### Build Time Configurations for r_slcdc

The following build time configurations are defined in fsp_cfg/r_slcdc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Graphics > Segment LCD (r_slcdc)**

This module can be added to the Stacks tab via New Stack > Graphics > Segment LCD (r_slcdc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_slcdc0 | Module Name |
| Clock | | | |
| Source | MCU Specific Options | | Select the clock source. |
| Divisor | MCU Specific Options | | Select the clock divisor. |
| Output | | | |
| Bias method | • 1/2 bias<br>• 1/3 bias<br>• 1/4 bias | 1/2 bias | Select the bias method. This determines the number of voltage levels used to create the waveforms. |
| Timeslice | MCU Specific Options | | Select the LCD time slice. The number of slices should match the number of common (COM) pins for your LCD panel. |
| Waveform | • Waveform A<br>• Waveform B | Waveform A | Select the LCD waveform. |
| Drive method | MCU Specific Options | | Select the LCD drive method. |
| Reference Voltage | MCU Specific Options | | Select the LCD reference voltage. |
| Default contrast (if available) | MCU Specific Options | | Select the default contrast level. |

**Valid Configurations**

Though there are many setting combinations only a limited subset are supported by the SLCDC peripheral hardware:

Boards have feature Select reference voltage mode (see relevant hardware manual for details)

| | | | | | | | |
|---|---|---|---|---|---|---|---|

| Waveform | Slices | Bias | External Resistance | Internal boosting VL1 | Internal boosting VL2 | Capacitor split VCC | Capacitor split VL4 |
|---|---|---|---|---|---|---|---|
| A | 8 | 1/4 | Available | Available | — | — | — |
| A | 6 | 1/4 | — | Available | — | — | — |
| A | 8 | 1/3 | Available | Available | Available | Available | Available |
| A | 6 | 1/3 | Available | Available | Available | Available | Available |
| A | 4 | 1/3 | Available | Available | Available | Available | Available |
| A | 3 | 1/3 | Available | Available | Available | Available | Available |
| A | 3 | 1/2 | Available | — | — | — | — |
| A | 2 | 1/2 | Available | — | — | — | — |
| A | Static | — | Available | — | — | — | — |
| B | 8 | 1/4 | Available | Available | — | — | — |
| B | 8 | 1/3 | Available | Available | Available | Available | Available |
| B | 6 | 1/3 | Available | Available | Available | Available | Available |
| B | 4 | 1/3 | Available | Available | Available | Available | Available |
| B | 3 | 1/3 | Available | Available | Available | Available | Available |

Others:

| Waveform | Slices | Bias | External Resistance | Internal Boost | Capacitor Split |
|---|---|---|---|---|---|
| A | 8 | 1/4 | Available | Available | — |
| A | 4 | 1/3 | Available | Available | Available |
| A | 3 | 1/3 | Available | Available | Available |
| A | 3 | 1/2 | Available | — | — |
| A | 2 | 1/2 | Available | — | — |
| A | Static | — | Available | — | — |
| B | 8 | 1/4 | Available | Available | Available |
| B | 4 | 1/3 | Available | Available | — |

### Clock Configuration

See relevant hardware manual: The SLCDC clock can be sourced from the main clock (MOSC), sub-clock (SOSC), HOCO, LOCO or MOCO. Dividers of 4 to 1024 are available for SOSC/LOCO and 256 to 1048576 are for MOSC/HOCO/MOCO. It is recommended to adjust the divisor such that the resulting clock provides a frame frequency of 32-128 Hz (some conditions have a frame frequency 24 to 128 Hz - see Table "Combinations of display waveform, time slices, bias method, and frame frequency" in the SLCDC section of the relevant hardware manual).

Others: The SLCDC clock can be sourced from the main clock (MOSC), sub-clock (SOSC), HOCO or LOCO. Dividers of 4 to 1024 are available for SOSC/LOCO and 256 to 524288 for MOSC/HOCO. It is recommended to adjust the divisor such that the resulting clock provides a frame frequency of 32-128 Hz.

*Note*

*Make sure your desired source clock is enabled and running before starting SLCDC output.*
*Do not set the segment LCD clock over 512 Hz when using internal boost or capacitor split modes.*

### Pin Configuration

This module controls a variety of pins necessary for segment LCD voltage generation and signal output:

| Pin Name | Function | Notes |
|---|---|---|
| SEGn | Segment data output | Connect these signals to the segment pins of the LCD. |
| COMn | Common signal output | Connect these signals to the common pins of the LCD. |
| VLn | Voltage reference | These pins should be connected to passive components based on the selected drive method (see "Supplying LCD Drive Voltages VL1, VL2, VL3, and VL4" in the SLCDC section of the relevant hardware manual). |
| CAPH, CAPL | Drive voltage generator capacitor | Connect a nonpolar 0.47uF capacitor across these pins when using internal boost or capacitor split modes. This pin is not needed when using resistance division. |

### Interrupt Configuration

The SLCDC provides no interrupt signals.

*Note*

*Blinking output timing is driven directly from the RTC periodic interrupt. Once the interrupt is enabled setting the display to SLCDC_DISP_BLINK will swap between A- and B-pattern each time it occurs. The ELC is not required for this functionality.*

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the SLCDC:

- Different packages provide different numbers of segment pins. Check the User's Manual for your device to confirm availability and mapping of segment signals.
- When using internal boost mode a delay of 5ms is required between calling R_SLCDC_Open and R_SLCDC_Start to allow the boost circuit to charge.

- When using the internal boost or capacitor split method do not set the segment LCD clock higher than 512 Hz.

# Examples

### Basic Example

Below is a basic example of minimal use of the SLCDC in an application. The SLCDC driver is initialized, output is started and a pattern is written to the segment registers.

```
void slcdc_init (void)
{
 fsp_err_t err;
 /* Open SLCDC driver */
    err = R_SLCDC_Open(&g_slcdc_ctrl, &g_slcdc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* When using internal boost mode this delay is required to allow the boost circuit
to charge. See
  * "Segment LCD Source Clock Control Register (SLCDSCKCR)" description in SLCDC
section of the relevant
  * hardware manual for details. */
 R_BSP_SoftwareDelay(5, BSP_DELAY_UNITS_MILLISECONDS);
 /* Start SLCDC output */
    err = R_SLCDC_Start(&g_slcdc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write pattern to display */
    err = R_SLCDC_Write(&g_slcdc_ctrl, 0, segment_data, NUM_SEGMENTS);
    assert(FSP_SUCCESS == err);
}
```

*Note*

*While the SLCDC is running, pattern data is constantly being output. No latching or buffering is required when writing or reading segment data.*

### Blinking Output

This example demonstrates how to set up blinking output using the RTC periodic interrupt. In this example it is assumed that the SLCDC has already been started.

```
void slcdc_blink (void)
{
 fsp_err_t err;
 /* Open RTC and set time/date */
    err = R_RTC_Open(&r_rtc_ctrl, &r_rtc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_RTC_CalendarTimeSet(&r_rtc_ctrl, &g_rtc_time);
    assert(FSP_SUCCESS == err);
 /* Set RTC periodic interrupt to 2 Hz (display blink cycle will be 1 Hz) */
    err = R_RTC_PeriodicIrqRateSet(&r_rtc_ctrl,
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND);
    assert(FSP_SUCCESS == err);
 /* Set display to blink */
    err = R_SLCDC_SetDisplayArea(&g_slcdc_ctrl, SLCDC_DISP_BLINK);
    assert(FSP_SUCCESS == err);
 /* Display will now continuously blink */
}
```

## Data Structures

| | |
|---|---|
| struct | slcdc_instance_ctrl_t |

## Data Structure Documentation

### ◆ slcdc_instance_ctrl_t

| struct slcdc_instance_ctrl_t |
|---|
| SLCDC control block. DO NOT INITIALIZE. Initialization occurs when slcdc_api_t::open is called |

## Function Documentation

### ◆ R_SLCDC_Open()

fsp_err_t R_SLCDC_Open ( slcdc_ctrl_t *const *p_ctrl*, slcdc_cfg_t const *const *p_cfg* )

Opens the SLCDC driver. Implements slcdc_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or the configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_UNSUPPORTED | Invalid display mode. |

### ◆ R_SLCDC_Write()

fsp_err_t R_SLCDC_Write ( slcdc_ctrl_t *const *p_ctrl*, uint8_t const *start_segment*, uint8_t const * *p_data*, uint8_t const *segment_count* )

Writes a sequence of display data to the segment data registers. Implements slcdc_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data was written successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block or data is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Segment index is (or will be) out of range. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized. |

### ◆ R_SLCDC_Modify()

fsp_err_t R_SLCDC_Modify ( slcdc_ctrl_t *const *p_ctrl*, uint8_t const *segment*, uint8_t const *data*, uint8_t const *data_mask* )

Modifies a single segment register based on a mask and the desired data. Implements slcdc_api_t::modify.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Invalid parameter in the argument. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_Start()

fsp_err_t R_SLCDC_Start ( slcdc_ctrl_t *const *p_ctrl*)

Starts output of LCD signals. Implements slcdc_api_t::start.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SLCDC timing, output, and voltage generation(if configured) activated |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_Stop()

fsp_err_t R_SLCDC_Stop ( slcdc_ctrl_t *const *p_ctrl*)

Stops output of LCD signals. Implements slcdc_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

### ◆ R_SLCDC_SetContrast()

fsp_err_t R_SLCDC_SetContrast ( slcdc_ctrl_t *const *p_ctrl*, slcdc_contrast_t const *contrast* )

Sets contrast to the specified level. Implements slcdc_api_t::setContrast.

***Note***

*Contrast can be adjusted when the SLCDC is operating in internal boost mode only. The range of values is 0-5 when 1/4 bias setting is used and 0-15 otherwise. See "LCD Boost Level Control Register (VLCD)" description in the SLCDC section of the relevant hardware manual for voltage levels at each setting.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |
| FSP_ERR_UNSUPPORTED | Unsupported operation |

### ◆ R_SLCDC_SetDisplayArea()

fsp_err_t R_SLCDC_SetDisplayArea ( slcdc_ctrl_t *const *p_ctrl*, slcdc_display_area_t const *display_area* )

Sets output to Waveform A, Waveform B or blinking output. Implements slcdc_api_t::setDisplayArea.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was opened successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_UNSUPPORTED | Pattern selection has no effect in 8-time-slice mode. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized. |

### ◆ R_SLCDC_Close()

fsp_err_t R_SLCDC_Close ( slcdc_ctrl_t *const *p_ctrl*)

Closes the SLCDC driver. Implements slcdc_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Device was closed successfully. |
| FSP_ERR_ASSERTION | Pointer to the control block structure is NULL. |
| FSP_ERR_NOT_OPEN | Device is not opened or initialized |

## 5.2.9 Input

Modules

### Detailed Description

Input Modules.

### Modules

| | |
|---|---|
| | External IRQ (r_icu)<br><br>Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface. |

| | |
|---|---|
| | Key Matrix (r_kint)<br><br>Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface. |

### 5.2.9.1 External IRQ (r_icu)

Modules » Input

**Functions**

| | |
|---|---|
| fsp_err_t | R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const p_api_ctrl, external_irq_cfg_t const *const p_cfg) |
| fsp_err_t | R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_ICU_ExternalIrqCallbackSet (external_irq_ctrl_t *const p_api_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_api_ctrl) |

**Detailed Description**

Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface.

# Overview

The Interrupt Controller Unit (ICU) controls which event signals are linked to the NVIC, DTC, and DMAC modules. The R_ICU software module only implements the External IRQ Interface. The external_irq interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

*Note*

*Multiple instances are used when more than one external interrupt is needed. Configure each instance with different channels and properties as needed for the specific interrupt.*

**Features**

- Supports configuring interrupts for IRQ pins on the target MCUs
  - Enabling and disabling interrupt generation.
  - Configuring interrupt trigger on rising edge, falling edge, both edges, or low level signal.
  - Enabling and disabling the IRQ noise filter.
- Supports configuring a user callback function, which will be invoked by the HAL module

when an external pin interrupt is generated.

# Configuration

### Build Time Configurations for r_icu

The following build time configurations are defined in fsp_cfg/r_icu_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Input > External IRQ (r_icu)

This module can be added to the Stacks tab via New Stack > Input > External IRQ (r_icu). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_external_irq0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the hardware channel. |
| Trigger | MCU Specific Options | | Select the signal edge or state that triggers an interrupt. |
| Digital Filtering | MCU Specific Options | | Select if the digital noise filter should be enabled. |
| Filter Source | MCU Specific Options | | Select the clock source for digital filter (Only affected when the digital filter is enabled). |
| Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled) | MCU Specific Options | | Select the clock divider for the digital noise filter. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers |

| | | |
|---|---|---|
| Pin Interrupt Priority | MCU Specific Options | Select the PIN interrupt priority. |

### Clock Configuration

The ICU peripheral module doesn't require any specific clock settings.

*Note*

*The digital filter uses PCLKB as the clock source for sampling the IRQ pin.*

### Pin Configuration

The pin for the external interrupt channel must be configured as an input with IRQ Input Enabled.

### Limitation

Pin configuration does not show conflicts when same IRQ is used by multiple pins.

## Usage Notes

### Digital Filter

The digital filter is used to reject trigger conditions that are too short. The trigger condition must be longer than three periods of the filter clock. The filter clock frequency is determined by PCLKB and the external_irq_pclk_div_t setting.

MIN_PULSE_WIDTH = EXTERNAL_IRQ_PCLKB_DIV / PCLKB_FREQUENCY * 3

### DMAC/DTC

When using an External IRQ pin to trigger a DMAC/DTC transfer, the External IRQ pin must be opened before the transfer instance is opened.

## Examples

### Basic Example

This is a basic example of minimal use of the ICU in an application.

```
#define ICU_IRQN_PIN BSP_IO_PORT_02_PIN_06
#define ICU_IRQN 6
/* Called from icu_irq_isr */
void external_irq_callback (external_irq_callback_args_t * p_args)
{
    (void) p_args;
    g_external_irq_complete = 1;
}
void simple_example ()
```

```
{
 /* Example Configuration */
 external_irq_cfg_t icu_cfg =
    {
        .channel          = ICU_IRQN,
        .trigger          = EXTERNAL_IRQ_TRIG_RISING,
        .filter_enable    = false,
        .clock_source_div = EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1,
        .p_callback       = external_irq_callback,
        .p_context       = 0,
        .ipl              = 0,
        .irq              = (IRQn_Type) 0,
    };
 /* Configure the external interrupt. */
 fsp_err_t err = R_ICU_ExternalIrqOpen(&g_icu_ctrl, &icu_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable the external interrupt. */
 /* Enable not required when used with ELC or DMAC. */
    err = R_ICU_ExternalIrqEnable(&g_icu_ctrl);
    assert(FSP_SUCCESS == err);
 while (0 == g_external_irq_complete)
    {
 /* Wait for interrupt. */
    }
}
```

### Data Structures

| | |
|---|---|
| struct | icu_extended_cfg_t |
| struct | icu_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | e_external_irq_digital_filter_t |

## Data Structure Documentation

### ◆ icu_extended_cfg_t

| struct icu_extended_cfg_t | | |
|---|---|---|
| Extended ICU interface configuration | | |
| Data Fields | | |
| e_external_irq_digital_filter_t | filter_src | Select digital filter clock source when digital filter is enabled. |

### ◆ icu_instance_ctrl_t

| struct icu_instance_ctrl_t | |
|---|---|
| ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if channel control block is in use. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number. |
| | |
| uint8_t | channel |
| | Channel. |
| | |
| void * | p_context |
| | |

## Field Documentation

### ◆ p_context

| void* icu_instance_ctrl_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in external_irq_callback_args_t. |

### Enumeration Type Documentation

### ◆ e_external_irq_digital_filter_t

| enum e_external_irq_digital_filter_t | |
|---|---|
| External IRQ input pin digital filtering clock selection. | |
| Enumerator | |
| EXTERNAL_IRQ_DIGITAL_FILTER_PCLK_DIV | Digital filter using clock PCLK dividers. |
| EXTERNAL_IRQ_DIGITAL_FILTER_LOCO | Digital filter using clock LOCO. |

## Function Documentation

### ◆ R_ICU_ExternalIrqOpen()

fsp_err_t R_ICU_ExternalIrqOpen ( external_irq_ctrl_t *const *p_api_ctrl*, external_irq_cfg_t const *const *p_cfg* )

Configure an IRQ input pin for use with the external interrupt interface. Implements external_irq_api_t::open.

The Open function is responsible for preparing an external IRQ pin for operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Open successful. |
| FSP_ERR_ASSERTION | One of the following is invalid:<br>• p_ctrl or p_cfg is NULL |
| FSP_ERR_ALREADY_OPEN | The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h. |
| FSP_ERR_INVALID_ARGUMENT | p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function. |
| FSP_ERR_UNSUPPORTED | An input argument is not supported by selected mode. |

***Note***

*This function is reentrant for different channels. It is not reentrant for the same channel.*

### ◆ R_ICU_ExternalIrqEnable()

fsp_err_t R_ICU_ExternalIrqEnable ( external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt for specified channel at NVIC. Implements external_irq_api_t::enable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt Enabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_IRQ_BSP_DISABLED | Requested IRQ is not defined in this system |

### ◆ R_ICU_ExternalIrqDisable()

fsp_err_t R_ICU_ExternalIrqDisable ( external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt for specified channel at NVIC. Implements external_irq_api_t::disable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt disabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |
| FSP_ERR_IRQ_BSP_DISABLED | Requested IRQ is not defined in this system |

### ◆ R_ICU_ExternalIrqCallbackSet()

fsp_err_t R_ICU_ExternalIrqCallbackSet ( external_irq_ctrl_t *const *p_api_ctrl*, void(*)(external_irq_callback_args_t *) *p_callback*, void *const *p_context*, external_irq_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements external_irq_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_ICU_ExternalIrqClose()

| fsp_err_t R_ICU_ExternalIrqClose ( external_irq_ctrl_t *const *p_api_ctrl*) |
|---|
| Close the external interrupt channel. Implements external_irq_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Successfully closed.<br>FSP_ERR_ASSERTION — The parameter p_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — The channel is not opened. |

## 5.2.9.2 Key Matrix (r_kint)

Modules » Input

### Functions

| | |
|---|---|
| fsp_err_t | R_KINT_Open (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_cfg_t const *const p_cfg) |
| fsp_err_t | R_KINT_Close (keymatrix_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_KINT_Enable (keymatrix_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_KINT_Disable (keymatrix_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface.

# Overview

The KINT module configures the Key Interrupt (KINT) peripheral to detect rising or falling edges on any of the KINT channels. When such an event is detected on any of the configured pins, the module generates an interrupt.

### Features

- Detect rising or falling edges on KINT channels
- Callback for notifying the application when edges are detected on the configured channels

# Configuration

### Build Time Configurations for r_kint

The following build time configurations are defined in fsp_cfg/r_kint_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Input > Key Matrix (r_kint)

This module can be added to the Stacks tab via New Stack > Input > Key Matrix (r_kint).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_kint0 | Module name. |
| Input | | | |
| Key Interrupt Flag Mask | MCU Specific Options | | Select channels to enable. |
| Interrupts | | | |
| Trigger Type | • Falling Edge<br>• Rising Edge | Rising Edge | Specifies if the enabled channels detect a rising edge or a falling edge. NOTE: either all channels detecting a rising edge or all channels detecting a falling edge. |
| Callback | Name must be a valid C symbol | kint_callback | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers. |
| Key Interrupt Priority | MCU Specific Options | | Select the key interrupt priority. |

### Clock Configuration

The KINT peripheral runs on PCLKB.

### Pin Configuration

The KRn pins are key switch matrix row input pins.

# Usage Notes

### Connecting a Switch Matrix

The KINT module is designed to scan the rows of a switch matrix where each row is connected to a number of columns through switches. A periodic timer (or other mechanism) sets one column pin high at a time. Any switches that are pressed on the driven column cause a rising (or falling) edge on the row pin (KRn) causing an interrupt.

*Note*

*In applications where multiple keys may be pressed at the same time it is recommended to put a diode inline with each switch to prevent ghosting.*

### Handling Multiple Pins

When an edge is detected on multiple pins at the same time, a single IRQ will be generated. A mask of all the pins that detected an edge will be passed to the callback.

# Examples

### Basic Example

This is a basic example of minimal use of the KINT in an application.

```
static volatile uint32_t g_channel_mask;
static volatile uint32_t g_kint_edge_detected = 0U;
/* Called from key_int_isr */
void r_kint_callback (keymatrix_callback_args_t * p_args)
{
    g_channel_mask       = p_args->channel_mask;
    g_kint_edge_detected = 1U;
}
void r_kint_example ()
{
 /* Configure the KINT. */
 fsp_err_t err = R_KINT_Open(&g_kint_ctrl, &g_kint_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the KINT. */
    err = R_KINT_Enable(&g_kint_ctrl);
    assert(FSP_SUCCESS == err);
 while (0 == g_kint_edge_detected)
    {
```

```
 /* Wait for interrupt. */
    }
}
```

### Data Structures

| | |
|---|---|
| struct | kint_instance_ctrl_t |

## Data Structure Documentation

### ◆ kint_instance_ctrl_t

| struct kint_instance_ctrl_t |
|---|
| Channel instance control block. DO NOT INITIALIZE. Initialization occurs when keymatrix_api_t::open is called. |

## Function Documentation

### ◆ R_KINT_Open()

| fsp_err_t R_KINT_Open ( keymatrix_ctrl_t *const *p_api_ctrl*, keymatrix_cfg_t const *const *p_cfg* ) |
|---|
| Configure all the Key Input (KINT) channels and provides a handle for use with the rest of the KINT API functions. Implements keymatrix_api_t::open. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | One of the following parameters may be NULL: p_cfg, or p_ctrl or the callback. |
| FSP_ERR_ALREADY_OPEN | The module has already been opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel mask is invalid. |

### ◆ R_KINT_Close()

| fsp_err_t R_KINT_Close ( keymatrix_ctrl_t *const *p_api_ctrl*) |
|---|
| Clear the KINT configuration and disable the KINT IRQ. Implements keymatrix_api_t::close. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The module is not opened. |

### ◆ R_KINT_Enable()

| fsp_err_t R_KINT_Enable ( keymatrix_ctrl_t *const *p_api_ctrl*) |
|---|
| This function enables interrupts for the KINT peripheral after clearing any pending requests. Implements keymatrix_api_t::enable. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt enabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The peripheral is not opened. |

### ◆ R_KINT_Disable()

| fsp_err_t R_KINT_Disable ( keymatrix_ctrl_t *const *p_api_ctrl*) |
|---|
| This function disables interrupts for the KINT peripheral. Implements keymatrix_api_t::disable. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Interrupt disabled successfully. |
| FSP_ERR_ASSERTION | The p_ctrl parameter was null. |
| FSP_ERR_NOT_OPEN | The channel is not opened. |

## 5.2.10 Monitoring

Modules

### Detailed Description

Monitoring Modules.

### Modules

| | |
|---|---|
| | CRC (r_crc)<br><br>Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface. |
| | Clock Accuracy Circuit (r_cac)<br><br>Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface. |

| | |
|---|---|
| | Data Operation Circuit (r_doc)<br>Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface. |
| | Independent Watchdog (r_iwdt)<br>Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface. |
| | Low/Programmable Voltage Detection (r_lvd)<br>Driver for the LVD and PVD peripherals on RA MCUs. This module implements the Low Voltage Detection Interface. |
| | Watchdog (r_wdt)<br>Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface. |

### 5.2.10.1 CRC (r_crc)

Modules » Monitoring

**Functions**

| | |
|---|---|
| fsp_err_t | R_CRC_Open (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg) |
| fsp_err_t | R_CRC_Close (crc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CRC_Calculate (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *calculatedValue) |
| fsp_err_t | R_CRC_CalculatedValueGet (crc_ctrl_t *const p_ctrl, uint32_t *calculatedValue) |
| fsp_err_t | R_CRC_SnoopEnable (crc_ctrl_t *const p_ctrl, uint32_t crc_seed) |
| fsp_err_t | R_CRC_SnoopDisable (crc_ctrl_t *const p_ctrl) |

**Detailed Description**

Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.

# Overview

The CRC module provides a API to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using industry-standard polynomials.

**Features**

- CRC module supports the following 8 and 16 bit CRC polynomials which operates on 8-bit data in parallel
  - X^8+X^2+X+1 (CRC-8)
  - X^16+X^15+X^2+1 (CRC-16)
  - X^16+X^12+X^5+1 (CRC-CCITT)
- CRC module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
  - X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X+ 1 (CRC-32)
  - X^32+ X^28+ X^27+ X^26+ X^25+ X^23+ X^22+ X^20+ X^19+ X^18+X^14+X^13+X^11+X^10+X^9+X^8+X^6+1 (CRC-32C)
- CRC module can calculate CRC with LSB first or MSB first bit order.

# Configuration

**Build Time Configurations for r_crc**

The following build time configurations are defined in fsp_cfg/r_crc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Monitoring > CRC (r_crc)**

This module can be added to the Stacks tab via New Stack > Monitoring > CRC (r_crc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_crc0 | Module name. |
| CRC Polynomial | MCU Specific Options | | Select the CRC polynomial. |
| Bit Order | MCU Specific Options | | Select the CRC bit order. |
| Snoop Address | MCU Specific Options | | Select the SCI register address CRC snoop |

**Clock Configuration**

There is no clock configuration for the CRC module.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### CRC Snoop

The CRC snoop function monitors reads from and writes to a specified I/O register address and performs CRC calculation on the data read from and written to the register address automatically. Instead of calling R_CRC_Calculate on a block of data, R_CRC_SnoopEnable is called to start monitoring reads/writes and R_CRC_CalculatedValueGet is used to obtain the current CRC.

*Note*

*Snoop mode is available for transmit/receive operations on SCI only.*

### Limitations

When using CRC32 polynomial functions the CRC module produces the same results as popular online CRC32 calculators, but it is important to remember a few important points.

- Online CRC32 calculators allow the input to be any number of bytes. The FSP CRC32 API function uses 32-bit words. This means the online calculations must be 'padded' to end on a 32-bit boundary.
- Online CRC32 calculators usually invert the output prior to presenting it as a result. It is up to the application program to include this step if needed.
- The seed value of 0xFFFFFFFF needs to be used by both the online calculator and the R_CRC module API (CRC32 polynomials)
- Make sure the bit orientation of the R_CRC CRC32 is set for LSB and that you have CRC32 selected and not CRC32C.
- Some online CRC tools XOR the final result with 0xFFFFFFFF.

# Examples

### Basic Example

This is a basic example of minimal use of the CRC module in an application.

```
void crc_example ()
{
    uint32_t length;
    uint32_t uint8_calculated_value;
    length = sizeof(g_data_8bit) / sizeof(g_data_8bit[0]);
 crc_input_t example_input =
    {
       .p_input_buffer = g_data_8bit,
       .num_bytes      = length,
```

```
      .crc_seed       = 0,
   };
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* 8-bit CRC calculation */
 R_CRC_Calculate(&crc_ctrl, &example_input, &uint8_calculated_value);
}
```

### Snoop Example

This example demonstrates CRC snoop operation.

```
void crc_snoop_example ()
{
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* Open SCI Driver */
 /* Configure Snoop address and enable snoop mode */
 R_CRC_SnoopEnable(&crc_ctrl, 0);
 /* Perfrom SCI read/Write operation depending on the SCI snoop address configure */
 /* Read CRC value */
 R_CRC_CalculatedValueGet(&crc_ctrl, &g_crc_buff);
}
```

## Data Structures

| | |
|---|---|
| struct | crc_instance_ctrl_t |

## Data Structure Documentation

### ◆ crc_instance_ctrl_t

| struct crc_instance_ctrl_t |
|---|
| Driver instance control structure. |

## Function Documentation

### ◆ R_CRC_Open()

fsp_err_t R_CRC_Open ( crc_ctrl_t *const *p_ctrl*, crc_cfg_t const *const *p_cfg* )

Open the CRC driver module

Implements crc_api_t::open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Module already open |
| FSP_ERR_UNSUPPORTED | Hardware not support this feature. |

### ◆ R_CRC_Close()

fsp_err_t R_CRC_Close ( crc_ctrl_t *const *p_ctrl*)

Close the CRC module driver.

Implements crc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_Calculate()

fsp_err_t R_CRC_Calculate ( crc_ctrl_t *const *p_ctrl*, crc_input_t *const *p_crc_input*, uint32_t * *calculatedValue* )

Perform a CRC calculation on a block of 8-bit/32-bit (for 32-bit polynomial) data.

Implements crc_api_t::calculate

This function performs a CRC calculation on an array of 8-bit/32-bit (for 32-bit polynomial) values and returns an 8-bit/32-bit (for 32-bit polynomial) calculated value

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calculation successful. |
| FSP_ERR_ASSERTION | Either p_ctrl, inputBuffer, or calculatedValue is NULL. |
| FSP_ERR_INVALID_ARGUMENT | length value is NULL, or not 4-byte aligned when 32-bit CRC polynomial function is configured. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_CalculatedValueGet()

fsp_err_t R_CRC_CalculatedValueGet ( crc_ctrl_t *const *p_ctrl*, uint32_t * *calculatedValue* )

Return the current calculated value.

Implements crc_api_t::crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Return of calculated value successful. |
| FSP_ERR_ASSERTION | Either p_ctrl or calculatedValue is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |

### ◆ R_CRC_SnoopEnable()

fsp_err_t R_CRC_SnoopEnable ( crc_ctrl_t *const *p_ctrl*, uint32_t *crc_seed* )

Configure the snoop channel and set the CRC seed.

Implements crc_api_t::snoopEnable

The CRC calculator can operate on reads and writes over any of the first ten SCI channels. For example, if set to channel 0, transmit, every byte written out SCI channel 0 is also sent to the CRC calculator as if the value was explicitly written directly to the CRC calculator.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snoop configured successfully. |
| FSP_ERR_ASSERTION | Pointer to control structure is NULL |
| FSP_ERR_NOT_OPEN | The driver is not opened. |
| FSP_ERR_UNSUPPORTED | SNOOP operation is not supported. |

### ◆ R_CRC_SnoopDisable()

fsp_err_t R_CRC_SnoopDisable ( crc_ctrl_t *const *p_ctrl*)

Disable snooping.

Implements crc_api_t::snoopDisable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Snoop disabled. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The driver is not opened. |
| FSP_ERR_UNSUPPORTED | SNOOP operation is not supported. |

## 5.2.10.2 Clock Accuracy Circuit (r_cac)

Modules » Monitoring

**Functions**

| | |
|---|---|
| fsp_err_t | R_CAC_Open (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg) |
| fsp_err_t | R_CAC_StartMeasurement (cac_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CAC_StopMeasurement (cac_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | R_CAC_Read (cac_ctrl_t *const p_ctrl, uint32_t *const p_counter) |
| fsp_err_t | R_CAC_CallbackSet (cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_CAC_Close (cac_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface.

# Overview

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of measurement clock edges that occur between two edges of the reference clock.

### Features

- Supports clock frequency-measurement and monitoring based on a reference signal input
- Reference can be either an externally supplied clock source or an internal clock source
- An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow.
- A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks.
- Edge-detection options for the reference clock are configurable as rising, falling, or both.

# Configuration

### Build Time Configurations for r_cac

The following build time configurations are defined in fsp_cfg/r_cac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Monitoring > Clock Accuracy Circuit (r_cac)

This module can be added to the Stacks tab via New Stack > Monitoring > Clock Accuracy Circuit (r_cac). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_cac0 | Module name. |

| | | | |
|---|---|---|---|
| Reference clock divider | • 32<br>• 128<br>• 1024<br>• 8192 | 32 | Reference clock divider. |
| Reference clock source | MCU Specific Options | | Reference clock source. |
| Reference clock digital filter | • Disabled<br>• Sampling clock =Measuring freq<br>• Sampling clock =Measuring freq/4<br>• Sampling clock =Measuring freq/16 | Disabled | Reference clock digital filter. |
| Reference clock edge detect | • Rising<br>• Falling<br>• Both | Rising | Reference clock edge detection. |
| Measurement clock divider | • 1<br>• 4<br>• 8<br>• 32 | 1 | Measurement clock divider. |
| Measurement clock source | MCU Specific Options | | Measurement clock source. |
| Upper Limit Threshold | Value must be a non-negative integer, between 0 to 65535 | 0 | Top end of allowable range for measurement completion. |
| Lower Limit Threshold | Value must be a non-negative integer, between 0 to 65535 | 0 | Bottom end of allowable range for measurement completion. |
| Frequency Error Interrupt Priority | MCU Specific Options | | CAC frequency error interrupt priority. |
| Measurement End Interrupt Priority | MCU Specific Options | | CAC measurement end interrupt priority. |
| Overflow Interrupt Priority | MCU Specific Options | | CAC overflow interrupt priority. |
| Callback | Name must be a valid C symbol | NULL | Function name for callback |

### Clock Configuration

The CAC measurement clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO

4. MOCO
5. LOCO
6. PCLKB
7. IWDT

The CAC reference clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT
8. External Clock Source (CACREF)

### Pin Configuration

The CACREF pin can be configured to provide the reference clock for CAC measurements.

## Usage Notes

### Measurement Accuracy

The clock measurement result may be off by up to one pulse depending on the phase difference between the edge detection circuit, digital filter, and CACREF pin signal, if applicable.

### Frequency Error Interrupt

The frequency error interrupt is only triggered at the end of a CAC measurement. This means that there will be a measurement complete interrupt in addition to the frequency error interrupt.

## Examples

### Basic Example

This is a basic example of minimal use of the CAC in an application.

```
volatile uint32_t g_callback_complete;
void cac_basic_example ()
{
    g_callback_complete = 0;
 fsp_err_t err = R_CAC_Open(&g_cac_ctrl, &g_cac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    (void) R_CAC_StartMeasurement(&g_cac_ctrl);
 /* Wait for measurement to complete. */
 while (0 == g_callback_complete)
```

```
    {
    }
    uint32_t value;
 /* Read the CAC measurement. */
    (void) R_CAC_Read(&g_cac_ctrl, &value);
}
/* Called when measurement is completed. */
static void r_cac_callback (cac_callback_args_t * p_args)
{
 if (CAC_EVENT_MEASUREMENT_COMPLETE == p_args->event)
    {
       g_callback_complete = 1U;
    }
}
```

### Data Structures

| | |
|---|---|
| struct | cac_instance_ctrl_t |

## Data Structure Documentation

### ◆ cac_instance_ctrl_t

| struct cac_instance_ctrl_t |
|---|
| CAC instance control block. DO NOT INITIALIZE. |

## Function Documentation

### ◆ R_CAC_Open()

| fsp_err_t R_CAC_Open ( cac_ctrl_t *const *p_ctrl*, cac_cfg_t const *const *p_cfg* ) |
|---|
| The Open function configures the CAC based on the provided user configuration settings. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC is available and available for measurement(s). |
| FSP_ERR_ASSERTION | An argument is invalid. |
| FSP_ERR_ALREADY_OPEN | The CAC has already been opened. |

***Note***

*There is only a single CAC peripheral.*

### ◆ R_CAC_StartMeasurement()

fsp_err_t R_CAC_StartMeasurement ( cac_ctrl_t *const *p_ctrl*)

Start the CAC measurement process.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC measurement started. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_StopMeasurement()

fsp_err_t R_CAC_StopMeasurement ( cac_ctrl_t *const *p_ctrl*)

Stop the CAC measurement process.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC measuring has been stopped. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_Read()

fsp_err_t R_CAC_Read ( cac_ctrl_t *const *p_ctrl*, uint32_t *const *p_counter* )

Read and return the CAC status and counter registers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CAC read successful. |
| FSP_ERR_ASSERTION | An argument is NULL. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

### ◆ R_CAC_CallbackSet()

fsp_err_t R_CAC_CallbackSet ( cac_ctrl_t *const *p_ctrl*, void(*)(cac_callback_args_t *) *p_callback*, void *const *p_context*, cac_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements cac_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_CAC_Close()

fsp_err_t R_CAC_Close ( cac_ctrl_t *const *p_ctrl*)

Release any resources that were allocated by the Open() or any subsequent CAC operations.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | NULL provided for p_instance_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | R_CAC_Open() has not been successfully called. |

## 5.2.10.3 Data Operation Circuit (r_doc)

Modules » Monitoring

### Functions

| | |
|---|---|
| fsp_err_t | R_DOC_Open (doc_ctrl_t *const p_api_ctrl, doc_cfg_t const *const p_cfg) |
| fsp_err_t | R_DOC_Close (doc_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DOC_Read (doc_ctrl_t *const p_api_ctrl, uint32_t *p_result) |
| fsp_err_t | R_DOC_Write (doc_ctrl_t *const p_api_ctrl, uint32_t data) |

| | |
|---|---|
| fsp_err_t | R_DOC_CallbackSet (doc_ctrl_t *const p_api_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface.

# Overview

### Features

The DOC HAL module peripheral is used to compare, add or subtract 16-bit or 32-bit[1] data and can detect the following events:

- Comparison Mode
  - Data is equal to the configured reference data setting.
  - Data is not equal to the configured reference data setting.
  - Data is less than the configured reference data setting[2].
  - Data is greater than the configured reference data setting[2].
  - Data is inside of a configurable pair of reference data settings[2].
  - Data is outside of a configurable pair of reference data settings[2].
- Addition Mode - Overflow of an addition operation
- Subtraction Mode - Underflow of a subtraction operation

A user-defined callback can be created to inform the CPU when any of above events occur.

*Note*

1. *Operating on 32-bit data is not supported on all MCUs.*
2. *This comparison mode is not supported on all MCUs.*

# Configuration

### Build Time Configurations for r_doc

The following build time configurations are defined in fsp_cfg/r_doc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Build Time Configurations for r_doc

The following build time configurations are defined in fsp_cfg/r_doc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled | Default (BSP) | If selected code for parameter checking is |

| | | | |
|---|---|---|---|
| | • Disabled | | included in the build. |

### Configurations for Monitoring > Data Operation Circuit (r_doc)

This module can be added to the Stacks tab via New Stack > Monitoring > Data Operation Circuit (r_doc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_doc0 | Module name. |
| Event | MCU Specific Options | | Select the event that will trigger the DOC interrupt. |
| Bit Width | MCU Specific Options | | The bit width for DOC operations. |
| Reference/Initial Data | Value must be an integer greater than or equal to 0. | 0 | Enter Initial Value for Addition/Subtraction or enter reference value for comparison. |
| Additional Reference Data | Value must be an integer greater than or equal to 0. | 0 | Additional reference data used for Window Compare modes. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided. This will be called from the interrupt service routine (ISR) when the configured DOC event occurs. |
| DOC Interrupt Priority | MCU Specific Options | | Select the DOC interrupt priority. |

### Configurations for Monitoring > Data Operation Circuit (r_doc)

This module can be added to the Stacks tab via New Stack > Monitoring > Data Operation Circuit (r_doc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_doc0 | Module name. |
| Event | MCU Specific Options | | Select the event that will trigger the DOC interrupt. |
| Bit Width | MCU Specific Options | | The bit width for DOC operations. |

| | | | |
|---|---|---|---|
| Reference/Initial Data | Value must be an integer greater than or equal to 0. | 0 | Enter Initial Value for Addition/Subtraction or enter reference value for comparison. |
| Additional Reference Data | Value must be an integer greater than or equal to 0. | 0 | Additional reference data used for Window Compare modes. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided. This will be called from the interrupt service routine (ISR) when the configured DOC event occurs. |
| DOC Interrupt Priority | MCU Specific Options | | Select the DOC interrupt priority. |

### Clock Configuration

The DOC HAL module does not require a specific clock configuration.

### Pin Configuration

The DOC HAL module does not require and specific pin configurations.

## Usage Notes

### DMAC/DTC Integration

DOC can be used with Transfer (r_dmac) or Transfer (r_dtc) to write to the input register without CPU intervention. DMAC is more useful for most DOC applications because it can be started directly from software. To write DOC input data with DTC/DMAC, set transfer_info_t::p_dest to R_DOC->DODIR.

## Examples

### Basic Example

This is a basic example of minimal use of the R_DOC in an application. This example shows how this driver can be used for continuous 16 bit addition operation while reading the result at every overflow event.

```
#define DOC_EXAMPLE_VALUE 0xF000

uint32_t g_callback_event_counter = 0;

/* This callback is called when DOC overflow event occurs. It is registered in

doc_cfg_t when R_DOC_Open is

 * called. */

void doc_callback (doc_callback_args_t * p_args)
```

```
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write data to the DOC Data Input Register and read the result of addition from
status register when an
  * interrupt occurs. */
 for (int i = 0; i < 5; i++)
    {
       err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
      assert(FSP_SUCCESS == err);
 if (g_callback_event_counter >= 1)
      {
            uint32_t result;
 /* Read the result of the operation */
            err = R_DOC_Read(&g_doc_ctrl, &result);
      assert(FSP_SUCCESS == err);
       }
    }
}
```

### Function Documentation

### ◆ R_DOC_Open()

fsp_err_t R_DOC_Open ( doc_ctrl_t *const *p_api_ctrl*, doc_cfg_t const *const *p_cfg* )

Opens and configures the Data Operation Circuit (DOC) in comparison, addition or subtraction mode and sets initial data for addition or subtraction, or reference data for comparison.

Example:

```
/* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | DOC successfully configured. |
| FSP_ERR_ALREADY_OPEN | Module already open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL or callback is NULL or the interrupt vector is invalid. |

### ◆ R_DOC_Close()

fsp_err_t R_DOC_Close ( doc_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | Pointer pointing to NULL. |

***Note***

*This function will disable the DOC interrupt in the NVIC.*

### ◆ R_DOC_Read()

fsp_err_t R_DOC_Read ( doc_ctrl_t *const *p_api_ctrl*, uint32_t * *p_result* )

Returns the result of addition/subtraction.

Example:

```
          uint32_t result;
/* Read the result of the operation */
          err = R_DOC_Read(&g_doc_ctrl, &result);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status successfully read. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL. |

### ◆ R_DOC_Write()

fsp_err_t R_DOC_Write ( doc_ctrl_t *const *p_api_ctrl*, uint32_t *data* )

Writes to the DODIR - DOC Input Register.

Example:

```
      err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Values successfully written to the registers. |
| FSP_ERR_NOT_OPEN | Driver not open. |
| FSP_ERR_ASSERTION | One or more pointers point to NULL. |

### ◆ R_DOC_CallbackSet()

fsp_err_t R_DOC_CallbackSet ( doc_ctrl_t *const *p_api_ctrl*, void(*)(doc_callback_args_t *) *p_callback*, void *const *p_context*, doc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements doc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.10.4 Independent Watchdog (r_iwdt)

Modules » Monitoring

### Functions

| | |
|---|---|
| fsp_err_t | R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg) |
| fsp_err_t | R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status) |
| fsp_err_t | R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status) |
| fsp_err_t | R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count) |
| fsp_err_t | R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout) |
| fsp_err_t | R_IWDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface.

# Overview

The independent watchdog timer is used to recover from unexpected errors in an application. The timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device or generates an NMI.

### Features

The IWDT HAL module has the following key features:

- When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an NMI
- The WDT supports following modes:[1]
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application[1].

### Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

| | WDT | IWDT |
|---|---|---|
| Start Mode | The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode). | On most of MCUs, the IWDT can only be configured by hardware to start automatically[1]. |
| Clock Source | The WDT runs off a peripheral clock. | The IWDT has its own clock source which improves safety. |

*Note*

*1. Refer to the MCU hardware user's manual or datasheet to determine if IWDT supports register start mode.*

# Configuration

When using register start mode, configure the watchdog timer on the Stacks tab.

*Note*

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab. These settings include the following:*

- *Start Mode*
- *Timeout Period*
- *Dedicated Clock Frequency Divisor*
- *Window End Position*
- *Window Start Position*
- *Reset Interrupt Request Select*
- *Stop Control*

*Review the OFS0 properties window to see additional details.*

### Build Time Configurations for r_iwdt

The following build time configurations are defined in fsp_cfg/r_iwdt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Register Start NMI Support | MCU Specific Options | | If enabled, code for NMI support in register start mode is included in the build. |

### Configurations for Monitoring > Independent Watchdog (r_iwdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Independent Watchdog (r_iwdt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_wdt0 | Module name. |
| Timeout | MCU Specific Options | | Select the independent watchdog timeout in cycles. |
| Clock Division Ratio | MCU Specific Options | | Select the independent watchdog clock divisor. |
| Window Start Position | MCU Specific Options | | Select the allowed independent watchdog refresh start point in %. |
| Window End Position | MCU Specific Options | | Select the allowed independent watchdog refresh end point in %. |
| Reset Control | MCU Specific Options | | Select what happens when the independent watchdog timer expires. |
| Stop Control | MCU Specific Options | | Select the independent watchdog state in low power mode. |
| NMI callback | Name must be a valid C symbol | NULL | A user callback function can be provided here. If this callback function is |

provided, it is called from the interrupt service routine (ISR) when the watchdog triggers.

### Clock Configuration

The IWDT clock is based on the IWDTCLK frequency. You can set the IWDTCLK frequency divider using the **BSP** tab of the RA Configuration editor.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### NMI Interrupt

The independent watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

*Note*

*When using the IWDT in software start mode with NMI and the timer underflows, the IWDT status must be reset by calling R_IWDT_StatusClear before restarting the timer via R_IWDT_Refresh.*

### Period Calculation

The IWDT operates from IWDTCLK. With a IWDTCLK of 15000 Hz, the maximum time from the last refresh to device reset or NMI generation will be just below 35 seconds as detailed below.

IWDTCLK = 15000 Hz
Clock division ratio = IWDTCLK / 256
Timeout period = 2048 cycles
WDT clock frequency = 15000 Hz / 256 = 58.59 Hz
Cycle time = 1 / 58.59 Hz = 17.067 ms
Timeout = 17.067 ms x 2048 cycles = 34.95 seconds

### Limitations

Developers should be aware of the following limitations when using the IWDT:

- When using a J-Link debugger the IWDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the IWDT to count and generate NMI or reset when the
 * debugger is connected. */
  R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
```

- If the IWDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_IWDT_Refresh() after the MCU wakes from low power mode.

# Examples

### IWDT Basic Example

This is a basic example of minimal use of the IWDT in an application.

```
void iwdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the IWDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
    }
}
```

### IWDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define IWDT_TIMEOUT_COUNTS (2048U)
#define IWDT_MAX_COUNTER (IWDT_TIMEOUT_COUNTS - 1U)
#define IWDT_START_WINDOW_75 ((IWDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void iwdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
```

```
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void iwdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the IWDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
#if defined(R_SYSTEM_RSTSR1_IWDTRF_Msk)
 /* (Optional) Check if the IWDTRF flag is set to know if the system is
  * recovering from a IWDT reset. */
 if (R_SYSTEM->RSTSR1_b.IWDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
#endif
 /* Open the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize other application code. */
 /* Do not call R_IWDT_Refresh() in auto start mode unless the
  * counter is in the acceptable refresh window. */
    (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

```
while (true)
   {
/* Application work here. */
/* (Optional) If there is a chance the application takes less time than
 * the start window, verify the IWDT counter is past the start window
 * before refreshing the IWDT. */
      uint32_t iwdt_counter = 0U;
do
     {
/* Read the current IWDT counter value. */
           err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     assert(FSP_SUCCESS == err);
     } while (iwdt_counter >= IWDT_START_WINDOW_75);
/* Refresh before the counter underflows to prevent reset or NMI. */
      (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
   }
}
```

## Data Structures

| | |
|---|---|
| struct | iwdt_instance_ctrl_t |

## Data Structure Documentation

### ◆ iwdt_instance_ctrl_t

| struct iwdt_instance_ctrl_t | |
|---|---|
| IWDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | wdt_open |
| | Indicates whether the open() API has been successfully called. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in wdt_callback_args_t. |
| | |
| void(* | p_callback )(wdt_callback_args_t *p_args) |

| | |
|---|---|
| | Callback provided when a WDT NMI ISR occurs. |
| | |

### Function Documentation

#### ◆ R_IWDT_Open()

fsp_err_t R_IWDT_Open ( wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg* )

Register the IWDT NMI callback.

Example:

```
/* Initializes the module. */
   err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT successfully configured. |
| FSP_ERR_ASSERTION | Null Pointer. |
| FSP_ERR_NOT_ENABLED | An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_STATE | The security state of the NMI and the module do not match. |

### ◆ R_IWDT_Refresh()

fsp_err_t R_IWDT_Refresh ( wdt_ctrl_t *const *p_api_ctrl*)

Refresh the Independent Watchdog Timer. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.

Example:

```
/* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
      (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT successfully refreshed. |
| FSP_ERR_ASSERTION | One or more parameters are NULL pointers. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_StatusGet()

fsp_err_t R_IWDT_StatusGet ( wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status* )

Read the IWDT status flags.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT status successfully read. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the IWDT generates an NMI when an error occurs. |

***Note***

*When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.*

### ◆ R_IWDT_StatusClear()

fsp_err_t R_IWDT_StatusClear ( wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status* )

Clear the IWDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT flag(s) successfully cleared. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the IWDT generates an NMI when an error occurs. |

***Note***

*When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.*

### ◆ R_IWDT_CounterGet()

fsp_err_t R_IWDT_CounterGet ( wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count* )

Read the current count value of the IWDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current IWDT counter value. */
           err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT current count successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_TimeoutGet()

fsp_err_t R_IWDT_TimeoutGet ( wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout* )

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | IWDT timeout information retrieved successfully. |
| FSP_ERR_ASSERTION | One or more parameters are NULL pointers. |
| FSP_ERR_NOT_OPEN | The driver has not been opened. Perform R_IWDT_Open() first. |

### ◆ R_IWDT_CallbackSet()

fsp_err_t R_IWDT_CallbackSet ( wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void *const *p_context*, wdt_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.10.5 Low/Programmable Voltage Detection (r_lvd)

Modules » Monitoring

### Functions

| | |
|---|---|
| fsp_err_t | R_LVD_Open (lvd_ctrl_t *const p_api_ctrl, lvd_cfg_t const *const p_cfg) |
| fsp_err_t | R_LVD_StatusGet (lvd_ctrl_t *const p_api_ctrl, lvd_status_t *p_lvd_status) |
| fsp_err_t | R_LVD_StatusClear (lvd_ctrl_t *const p_api_ctrl) |

| | |
|---|---|
| fsp_err_t | R_LVD_CallbackSet (lvd_ctrl_t *const p_api_ctrl, void(*p_callback)(lvd_callback_args_t *), void *const p_context, lvd_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_LVD_Close (lvd_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the LVD and PVD peripherals on RA MCUs. This module implements the Low Voltage Detection Interface.

*Note*

*In the below usage notes, "LVD" refers to both LVD and PVD.*

# Overview

The Low Voltage Detection module configures the voltage monitors to detect when a power supply pin or a voltage detector pin voltages crosses a specified threshold.

### Features

The LVD HAL module supports the following functions:

- Seven run-time configurable voltage monitors (Voltage Monitor 1, Voltage Monitor 2, Voltage Monitor 4, Voltage Monitor 5, LVD VBAT, LVD VRTC, EXLVD)
  - Configurable voltage threshold
  - Digital filter (Available on specific MCUs)
  - Support for both interrupt or polling
    - NMI or maskable interrupt can be configured (NMI support for Voltage Monitor 1 & Voltage Monitor 2 only).
  - Voltage monitor interrupt generation condition can be configured (rising, falling, or both edge event detection).
  - Support for resetting the MCU when $V_{CC}$ falls below/rises above configured threshold.
    **Note**: Voltage monitor availability is limited based on the selected MCU.

# Configuration

### Build Time Configurations for r_lvd

The following build time configurations are defined in fsp_cfg/r_lvd_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Monitoring > Low/Programmable Voltage Detection (r_lvd)

This module can be added to the Stacks tab via New Stack > Monitoring > Low/Programmable Voltage Detection (r_lvd). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_lvd0 | Module name. |
| Monitor Number | MCU Specific Options | | Select the LVD monitor. |
| Digital Filter | MCU Specific Options | | Enable the digital filter and select the digital filter clock divider. |
| Voltage Threshold | MCU Specific Options | | Select the low voltage detection threshold. |
| Detection Response | MCU Specific Options | | Select what happens when the voltage crosses the threshold voltage. |
| Voltage Slope | MCU Specific Options | | Select interrupt generation on rising voltage, falling voltage or both. |
| Negation Delay | MCU Specific Options | | Negation of the monitor signal can either be delayed from the reset event or from voltage returning to normal range. |
| Monitor Interrupt Callback | Name must be a valid C symbol. | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers. |
| LVD Monitor Interrupt Priority | MCU Specific Options | | Select the LVD Monitor interrupt priority. |

**Clock Configuration**

The LOCO clock must be enabled in order to use the digital filter.

**Pin Configuration**

To use LVD module, you need to switch pin function of EXLVDVBAT pin, VRTC pin, EXLVD pin to enable LVD function on this pin.

# Usage Notes

### Startup Edge Detection

If $V_{CC}$ is below the threshold prior to configuring the voltage monitor for falling edge detection, the monitor will immediately detect the a falling edge condition. If $V_{CC}$ is above the threshold prior to configuring the monitor for rising edge detection, the monitor will not detect a rising edge condition until $V_{CC}$ falls below the threshold and then rises above it again.

### Voltage Monitor 0

The LVD HAL module only supports configuring voltage monitor 1 and voltage monitor 2. Voltage monitor 0 can be configured by setting the appropriate bits in the OFS1 register. This means that voltage monitor 0 settings cannot be changed at runtime.

Voltage monitor 0 supports the following features

- Configurable Voltage Threshold ($V_{DET0}$)
- Reset the device when $V_{CC}$ falls below $V_{DET0}$

### Limitations

- The digital filter must be disabled when using voltage monitors in Software Standby or Deep Software Standby.
- Deep Software Standby mode is not possible if the voltage monitor is configured to reset the MCU.
- Reset generated by the VCC-rise detection is supported only on devices with PVD.
- On RA0, when LVD0 is used the detection voltage of LVD1 must be higher than LVD0's. See in hardware manual: LVD > Register Descriptions > LVD1CR.
- Voltage Monitor 4, Voltage Monitor 5 does not support interrupt feature.
- LVD Monitor 1 and LVD Monitor 2 must have the same security attribution. Either both monitors are used in the secure project, or both monitors are used in the non-secure project.

# Examples

### Basic Example

This is a basic example of minimal use of the LVD in an application.

```
void basic_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 lvd_status_t status;
       err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
      assert(FSP_SUCCESS == err);
```

```
 if (LVD_THRESHOLD_CROSSING_DETECTED == status.crossing_detected)
      {
            err = R_LVD_StatusClear(&g_lvd_ctrl);
      assert(FSP_SUCCESS == err);
 /* Do something */
       }
    }
}
```

### Interrupt Example

This is a basic example of using a LVD instance that is configured to generate an interrupt.

```
void interrupt_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Application Process */
 /* Application will be interrupted when Vcc crosses the configured threshold. */
    }
}
/* Called when Vcc crosses configured threshold. */
void lvd_callback (lvd_callback_args_t * p_args)
{
 if (LVD_CURRENT_STATE_BELOW_THRESHOLD == p_args->current_state)
    {
 /* Do Something */
    }
}
```

### Reset Example

This is a basic example of using a LVD instance that is configured to reset the MCU.

```
void reset_example (void)
{
 if (1U == R_SYSTEM->RSTSR0_b.LVD1RF)
    {
 /* The system is coming out of reset because Vcc crossed configured voltage
threshold. */
 /* Clear Voltage Monitor 1 Reset Detect Flag. */
       R_SYSTEM->RSTSR0_b.LVD1RF = 0;
    }
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Application Process */
 /* Application will reset when Vcc crosses the configured threshold. */
    }
}
```

## Data Structures

| | |
|---|---|
| struct | lvd_instance_ctrl_t |

## Data Structure Documentation

### ◆ lvd_instance_ctrl_t

| |
|---|
| struct lvd_instance_ctrl_t |
| LVD instance control structure |

## Function Documentation

### ◆ R_LVD_Open()

fsp_err_t R_LVD_Open ( lvd_ctrl_t *const *p_api_ctrl*, lvd_cfg_t const *const *p_cfg* )

Initializes a voltage monitor and detector according to the passed-in configuration structure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |
| [in] | p_cfg | Pointer to the configuration structure for the driver instance |

***Note***

*Digital filter is not to be used with standby modes.*
*Startup time can take on the order of milliseconds for some configurations.*

Example:

```
fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | Requested configuration was invalid |
| FSP_ERR_ALREADY_OPEN | The instance was already opened |
| FSP_ERR_IN_USE | Another instance is already using the desired monitor |
| FSP_ERR_UNSUPPORTED | Digital filter was enabled on a device that does not support it |

### ◆ R_LVD_StatusGet()

fsp_err_t R_LVD_StatusGet ( lvd_ctrl_t *const *p_api_ctrl*, lvd_status_t * *p_lvd_status* )

Get the current state of the monitor (threshold crossing detected, voltage currently above or below threshold).

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |
| [out] | p_lvd_status | Pointer to status structure |

Example:

```
err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |
| FSP_ERR_UNSUPPORTED | This monitor does not support status feature |

### ◆ R_LVD_StatusClear()

fsp_err_t R_LVD_StatusClear ( lvd_ctrl_t *const *p_api_ctrl*)

Clears the latched status of the monitor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control structure for the driver instance |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |
| FSP_ERR_UNSUPPORTED | This monitor does not support status feature |

### ◆ R_LVD_CallbackSet()

fsp_err_t R_LVD_CallbackSet ( lvd_ctrl_t *const *p_api_ctrl*, void(*)(lvd_callback_args_t *) *p_callback*, void *const *p_context*, lvd_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements lvd_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_LVD_Close()

fsp_err_t R_LVD_Close ( lvd_ctrl_t *const *p_api_ctrl*)

Disables the LVD peripheral. Closes the driver instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to the control block structure for the driver instance |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_NOT_OPEN | Driver is not open |

## 5.2.10.6 Watchdog (r_wdt)

Modules » Monitoring

**Functions**

| | |
|---|---|
| fsp_err_t | R_WDT_Open (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg) |
| fsp_err_t | R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout) |

| | |
|---|---|
| fsp_err_t | R_WDT_Refresh (wdt_ctrl_t *const p_ctrl) |
| fsp_err_t | R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status) |
| fsp_err_t | R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl, const wdt_status_t status) |
| fsp_err_t | R_WDT_CounterGet (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count) |
| fsp_err_t | R_WDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface.

# Overview

The watchdog timer is used to recover from unexpected errors in an application. The watchdog timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an NMI.

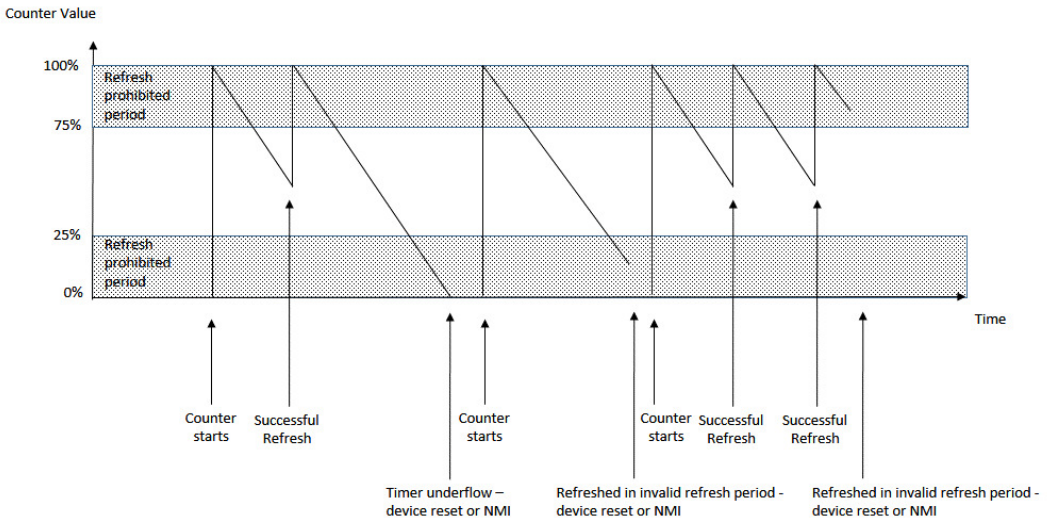



Figure 260: Watchdog Timer Operation Example

### Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an NMI
- The WDT has two supported modes:
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application.

### Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

| | WDT | IWDT |
|---|---|---|
| Start Mode | The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode). | The IWDT can only be configured by hardware to start automatically. |
| Clock Source | The WDT runs off a peripheral clock. | The IWDT has its own clock source which improves safety. |

## Configuration

When using register start mode, configure the watchdog timer on the Stacks tab.

*Note*

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab.*

### Build Time Configurations for r_wdt

The following build time configurations are defined in fsp_cfg/r_wdt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Register Start NMI Support | • Enabled<br>• Disabled | Disabled | If enabled, code for NMI support in register start mode is included in the build. |

### Configurations for Monitoring > Watchdog (r_wdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Watchdog (r_wdt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_wdt0 | Module name. |
| Timeout | • 1,024 Cycles<br>• 4,096 Cycles<br>• 8,192 Cycles<br>• 16,384 Cycles | 16,384 Cycles | Select the watchdog timeout in cycles. |
| Clock Division Ratio | • PCLK/4<br>• PCLK/64<br>• PCLK/128<br>• PCLK/512<br>• PCLK/2048<br>• PCLK/8192 | PCLK/8192 | Select the watchdog clock divisor. |
| Window Start Position | • 100 (Window Position Not Specified)<br>• 75<br>• 50<br>• 25 | 100 (Window Position Not Specified) | Select the allowed watchdog refresh start point in %. |
| Window End Position | • 0 (Window Position Not Specified)<br>• 25<br>• 50<br>• 75 | 0 (Window Position Not Specified) | Select the allowed watchdog refresh end point in %. |
| Reset Control | • Reset Output<br>• NMI Generated | Reset Output | Select what happens when the watchdog timer expires. |
| Stop Control | • WDT Count Enabled in Low Power Mode<br>• WDT Count Disabled in Low Power Mode | WDT Count Disabled in Low Power Mode | Select the watchdog state in low power mode. |
| NMI Callback | Name must be a valid C symbol | NULL | A user callback function must be provided if the WDT is configured to generate an NMI when the timer underflows or a refresh error occurs. If this callback function is provided, it will be called from the NMI handler each time the |

watchdog triggers.

### Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### NMI Interrupt

The watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

*Note*

*When using the WDT in software start mode with NMI and the timer underflows, the WDT status must be reset by calling R_WDT_StatusClear before restarting the timer via R_WDT_Refresh.*

### Period Calculation

The WDT operates from PCLKB. With a PCLKB of 60 MHz, the maximum time from the last refresh to device reset or NMI generation will be just over 2.2 seconds as detailed below.

PLCKB = 60 MHz
Clock division ratio = PCLKB / 8192
Timeout period = 16384 cycles
WDT clock frequency = 60 MHz / 8192 = 7.324 kHz
Cycle time = 1 / 7.324 kHz = 136.53 us
Timeout = 136.53 us x 16384 cycles = 2.23 seconds

### Limitations

Developers should be aware of the following limitations when using the WDT:

- When using a J-Link debugger the WDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the WDT to count and generate NMI or reset when the
 * debugger is connected. */
  R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
```

- If the WDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_WDT_Refresh() after the MCU wakes from low power mode.

## Examples

### WDT Basic Example

This is a basic example of minimal use of the WDT in an application.

```
void wdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the WDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* In register start mode, start the watchdog by calling R_WDT_Refresh. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset or NMI. */
      err = R_WDT_Refresh(&g_wdt0_ctrl);
      assert(FSP_SUCCESS == err);
    }
}
```

### WDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define WDT_TIMEOUT_COUNTS (16384U)
#define WDT_MAX_COUNTER (WDT_TIMEOUT_COUNTS - 1U)
#define WDT_START_WINDOW_75 ((WDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void wdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
    assert(FSP_SUCCESS == err);
 /* (Register start mode) In register start mode, call R_WDT_Refresh() to
  * continue using the watchdog after an error. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void wdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the WDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
 /* (Optional) Check if the WDTRF flag is set to know if the system is
  * recovering from a WDT reset. */
 if (R_SYSTEM->RSTSR1_b.WDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
 /* Open the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize other application code. */
```

```
/* (Register start mode) Call R_WDT_Refresh() to start the WDT in register
 * start mode. Do not call R_WDT_Refresh() in auto start mode unless the
 * counter is in the acceptable refresh window. */
   err = R_WDT_Refresh(&g_wdt0_ctrl);
   assert(FSP_SUCCESS == err);
while (true)
   {
/* Application work here. */
/* (Optional) If there is a chance the application takes less time than
 * the start window, verify the WDT counter is past the start window
 * before refreshing the WDT. */
      uint32_t wdt_counter = 0U;
do
     {
/* Read the current WDT counter value. */
           err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
     assert(FSP_SUCCESS == err);
     } while (wdt_counter >= WDT_START_WINDOW_75);
/* Refresh before the counter underflows to prevent reset or NMI. */
     err = R_WDT_Refresh(&g_wdt0_ctrl);
     assert(FSP_SUCCESS == err);
   }
}
```

## Data Structures

| | |
|---|---|
| struct | wdt_instance_ctrl_t |

## Data Structure Documentation

### ◆ wdt_instance_ctrl_t

| struct wdt_instance_ctrl_t |
|---|
| WDT private control block. DO NOT MODIFY. Initialization occurs when R_WDT_Open() is called. |

## Function Documentation

### ◆ R_WDT_Open()

fsp_err_t R_WDT_Open ( wdt_ctrl_t *const *p_ctrl*, wdt_cfg_t const *const *p_cfg* )

Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements wdt_api_t::open.

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_STATE | The security state of the NMI and the module do not match. |

***Note***

*In auto start mode the only valid configuration option is for registering the callback for the NMI ISR if NMI output has been selected.*

### ◆ R_WDT_TimeoutGet()

fsp_err_t R_WDT_TimeoutGet ( wdt_ctrl_t *const *p_ctrl*, wdt_timeout_values_t *const *p_timeout* )

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT timeout information retrieved successfully. |
| FSP_ERR_ASSERTION | Null Pointer. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

### ◆ R_WDT_Refresh()

fsp_err_t R_WDT_Refresh ( wdt_ctrl_t *const *p_ctrl*)

Refresh the watchdog timer. Implements wdt_api_t::refresh.

In addition to refreshing the watchdog counter this function can be used to start the counter in register start mode.

Example:

```
/* Refresh before the counter underflows to prevent reset or NMI. */
     err = R_WDT_Refresh(&g_wdt0_ctrl);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT successfully refreshed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

***Note***

*This function only returns FSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.*

### ◆ R_WDT_StatusGet()

fsp_err_t R_WDT_StatusGet ( wdt_ctrl_t *const *p_ctrl*, wdt_status_t *const *p_status* )

Read the WDT status flags. Implements wdt_api_t::statusGet.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT status successfully read. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the watchdog generates an NMI when an error occurs. |

***Note***

*When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.*

### ◆ R_WDT_StatusClear()

fsp_err_t R_WDT_StatusClear ( wdt_ctrl_t *const *p_ctrl*, const wdt_status_t *status* )

Clear the WDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT flag(s) successfully cleared. |
| FSP_ERR_ASSERTION | Null pointer as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |
| FSP_ERR_UNSUPPORTED | This function is only valid if the watchdog generates an NMI when an error occurs. |

***Note***

*When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.*

### ◆ R_WDT_CounterGet()

fsp_err_t R_WDT_CounterGet ( wdt_ctrl_t *const *p_ctrl*, uint32_t *const *p_count* )

Read the current count value of the WDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current WDT counter value. */
           err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WDT current count successfully read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Instance control block is not initialized. |

### ◆ R_WDT_CallbackSet()

fsp_err_t R_WDT_CallbackSet ( wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void *const *p_context*, wdt_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.11 Motor

Modules

### Detailed Description

Motor Modules.

### Modules

| | |
|---|---|
| | 120-degree conduction control sensorless (rm_motor_120_control_sensorless)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Control Interface. |
| | 120-degree conduction control with Hall sensors (rm_motor_120_control_hall)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Control Interface. |
| | ADC and PWM Modulation (rm_motor_driver)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor driver Interface. |

| | |
|---|---|
| | ADC and PWM modulation (rm_motor_120_driver)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Driver Interface. |
| | Motor 120 degree control (rm_motor_120_degree)<br><br>Usual control of a SPM (Surface Permanent Magnet) motor on RA MCUs. This module implements the Motor 120 degree control (rm_motor_120_degree). |
| | Motor Angle (rm_motor_estimate)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface. |
| | Motor Angle (rm_motor_sense_encoder)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface. |
| | Motor Angle and Speed Calculation with Hall sensors (rm_motor_sense_hall)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface. |
| | Motor Angle and Speed Calculation with induction sensor (rm_motor_sense_induction)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface. |
| | Motor Current Controller (rm_motor_current)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor current Interface. |
| | Motor Encoder Vector Control (rm_motor_encoder)<br><br>Control a SPM motor on RA MCUs. This module implements the Motor Encoder Vector Control (rm_motor_encoder). |
| | Motor Inertia estimate (rm_motor_inertia_estimate)<br><br>Measurement and calculation process for the motor control on RA MCUs. This module implements the Motor Inertia Estimate Interface. |

| | |
|---|---|
| | Motor Position Controller (rm_motor_position)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor position Interface. |
| | Motor Sensorless Vector Control (rm_motor_sensorless)<br><br>Usual control of a SPM motor on RA MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless). |
| | Motor Speed Controller (rm_motor_speed)<br><br>Calculation process for the motor control on RA MCUs. This module implements the Motor speed Interface. |
| | Motor Vector Control with hall sensors (rm_motor_hall)<br><br>Usual control of a SPM motor on RA MCUs. This module implements the Motor Vector Control with hall sensors (rm_motor_hall). |
| | Motor return origin (rm_motor_return_origin)<br><br>Search origin position process for the motor control on RA MCUs. This module implements the Motor Return Origin Function Interface. |
| | Motor vector control with induction sensor (rm_motor_induction)<br><br>Control a SPM motor on RA MCUs. This module implements the Motor vector control with induction sensor (rm_motor_induction). |

### 5.2.11.1 120-degree conduction control sensorless (rm_motor_120_control_sensorless)

Modules » Motor

#### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_Open (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)<br><br>Opens and configures the motor sensorless 120 detection module. Implements motor_120_control_api_t::open. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_Close (motor_120_control_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | Disables specified motor sensorless 120 detection module. Implements motor_120_control_api_t::close. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_Run (motor_120_control_ctrl_t *const p_ctrl)<br>Run motor (Start motor rotation). Implements motor_120_control_api_t::run. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_Stop (motor_120_control_ctrl_t *const p_ctrl)<br>Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_Reset (motor_120_control_ctrl_t *const p_ctrl)<br>Reset variables of motor sensorless 120 detection module. Implements motor_120_control_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed[rpm]. Implements motor_120_control_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get speed. Implements motor_120_control_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)<br>Get current. Implements motor_120_control_api_t::currentGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag)<br>Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet |

| | |
|---|---|
| | (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag)<br><br>Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag)<br><br>Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref)<br><br>Get voltage ref. Implements motor_120_control_api_t::voltageRefGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)<br><br>Update the parameters of sensorless 120 detection module. Implements motor_120_control_api_t::parameterUpdate. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Control Interface.

# Overview

The motor current is used to control the electric current of motor rotation in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

### Features

The motor 120 control sensorless module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling control.
- Voltage error compensation.

# Configuration

### Build Time Configurations for rm_motor_120_control_sensorless

The following build time configurations are defined in fsp_cfg/rm_motor_120_control_sensorless_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > 120-degree conduction control sensorless (rm_motor_120_control_sensorless)

This module can be added to the Stacks tab via New Stack > Motor > 120-degree conduction control sensorless (rm_motor_120_control_sensorless).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_120_control_sensorless0 | Module name. |
| Conduction type | • First 60 degree PWM<br>• Complementary First 60 degree PWM | First 60 degree PWM | Select conduction type |
| Stop BEMF | Must be a valid non-negative value. | 0.5 | Value of stop motor BEMF (U+V+W) |
| Timeout counts (msec) | Must be a valid integer. | 2000 | Undetected time |
| Maximum voltage for BOOT (V) | Must be a valid non-negative value. | 8.0 | Maximum output voltage for boot mode (V) |
| Maximum voltage (V) | Must be a valid non-negative value. | 20.0 | Maximum output voltage (V) |
| Minimum voltage (V) | Must be a valid non-negative value. | 3.0 | Minimum output voltage (V) |
| Carrier frequency (kHz) | Must be a valid non-negative value. | 20.0 | PWM carrier frequency (kHz) |
| Adjusting angle | Must be a valid non-negative value. | 0 | Adjusting angle |
| Speed PI decimation | Must be a valid integer. | 1 | Speed PI control decimation count |
| Free run timer frequency (MHz) | Must be a valid integer. | 120 | Freerun timer frequency (MHz) |
| Speed LPF K | Must be a valid non-negative value. | 1.0 | Speed LPF parameter |

| | | | |
|---|---|---|---|
| Step of speed change | Must be a valid non-negative value. | 0.2 | Speed reference change step |
| Boot reference voltage (V) | Must be a valid non-negative value. | 3.0 | Voltage reference for boot mode |
| Voltage lamping time | Must be a valid integer. | 128 | Voltage lamping time |
| Voltage constant adjust time | Must be a valid integer. | 64 | Voltage constant adjust time value (msec) |
| Open loop start speed (rpm) | Must be a valid integer value. | 150 | Open loop start speed (rpm) |
| Open loop mode2 speed (rpm) | Must be a valid integer value. | 185 | to mode2 change speed (rpm) |
| Open loop mode3 speed (rpm) | Must be a valid integer value. | 1000 | to mode3 change speed (rpm) |
| Open loop start voltage (V) | Must be a valid non-negative value. | 3.0 | start reference voltage (V) |
| Open loop mode1 speed rate | Must be a valid non-negative value. | 0.25 | increase rate of reference speed (rpm/control period) |
| Open loop mode2 voltage rate | Must be a valid non-negative value. | 0.00285 | increase rate of reference voltage (v/control period) |
| Open loop mode2 speed rate | Must be a valid non-negative value. | 0.71 | increase rate of reference speed (rpm/control period) |
| Open loop mode3 voltage rate | Must be a valid non-negative value. | 0.002 | increase rate of reference voltage (v/control period) |
| Open loop maximum voltage (V) | Must be a valid non-negative value. | 6.5 | openloop maximum voltage (V) |
| PI control KP | Must be a valid non-negative value. | 0.02 | PI control gain of proportional term |
| PI control KI | Must be a valid non-negative value. | 0.004 | PI control gain of integral term |
| PI control limit | Must be a valid non-negative value. | 24.0 | PI control limit of integral term |
| Motor Parameter | | | |
| Pole pairs | Must be a valid integer. | 2 | Pole pairs |
| Resistance (ohm) | Must be a valid non-negative value. | 6.447 | Resistance |
| Inductance of d-axis (H) | Must be a valid non-negative value. | 0.0045 | Inductance of d-axis |
| Inductance of q-axis | Must be a valid non- | 0.0045 | Inductance of q-axis |

| | | | |
|---|---|---|---|
| (H) | negative value. | | |
| Permanent magnetic flux (Wb) | Must be a valid non-negative value. | 0.02159 | Permanent magnetic flux |
| Motor Parameter > Rotor inertia (kgm^2) | Must be a valid non-negative value. | 1.8 | Rotor inertia |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | callback function |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Limitations

- Set the period of current control with none-negative value.
- Set the reference voltage with none-negative value.

## Examples

### Basic Example

This is a basic example of minimal use of the motor 120 control sensorless in an application.

```
void motor_120_control_sensorless_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err =
RM_MOTOR_120_CONTROL_SENSORLESS_Open(g_motor_120_control_sensorless0.p_ctrl,
                                          g_motor_120_control_sensorless0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet(g_motor_120_control_sensorless0.p_ctrl,

DEF_120_CONTROL_SENSORLESSHALL_TEST_OVSPD_LIM);
```

```
 /* Start motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Run(g_motor_120_control_sensorless0.p_ctrl);
 /* Get current motor speed */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_speed);
 /* Get current */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_current_status);
 /* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                    &smpl_wait_stop_flag);
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                         &smpl_timeout_error_fl
ag);
 /* Get pattern error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                         &smpl_pattern_error_fl
ag);
 /* Get voltage ref */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_voltage_ref);
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate
```

```
(g_motor_120_control_sensorless0.p_ctrl,

g_motor_120_control_sensorless0.p_cfg);
 /* Stop motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Stop(g_motor_120_control_sensorless0.p_ctrl);
 /* Reset the process. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Reset(g_motor_120_control_sensorless0.p_ctrl);
 /* Close the module. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Close(g_motor_120_control_sensorless0.p_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | motor_120_control_sensorless_extended_cfg_t |
| struct | motor_120_control_sensorless_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_120_control_sensorless_draw_in_position_t |
| enum | motor_120_control_sensorless_pattern_change_flag_t |

## Data Structure Documentation

### ◆ motor_120_control_sensorless_extended_cfg_t

| struct motor_120_control_sensorless_extended_cfg_t | | |
|---|---|---|
| Extended configurations for motor 120 control sensorless | | |
| Data Fields | | |
| float | f4_stop_bemf | Value of stop motor BEMF (U+V+W) |
| float | f4_max_boot_v | Max output voltage for boot mode (V) |
| float | f4_carrier_freq | Carrier wave frequency (kHz) |
| int32_t | s4_angle_shift_adjust | Adjusting angle. |
| float | f4_boot_ref_v | Voltage reference when zero speed (V) |
| | | |

| | | |
|---|---|---|
| uint32_t | u4_v_up_time | Voltage lamping time. |
| uint32_t | u4_v_const_time | Voltage constant adjust time value (ms) |
| int32_t | s4_ol_start_rpm | Start speed (rpm) |
| int32_t | s4_ol_mode1_change_rpm | To mode2 change speed (rpm) |
| int32_t | s4_ol_mode2_change_rpm | To mode3 change speed (rpm) |
| float | f4_ol_start_refv | Start reference voltage (V) |
| float | f4_ol_mode1_rate_rpm | Increase rate of reference speed (rpm/control period) |
| float | f4_ol_mode2_rate_refv | Increase rate of reference voltage (v/control period) |
| float | f4_ol_mode2_rate_rpm | Increase rate of reference speed (rpm/control period) |
| float | f4_ol_mode3_rate_refv | Increase rate of reference voltage (v/control period) |
| float | f4_ol_mode3_max_refv | Openloop max voltage (V) |
| motor_120_driver_instance_t const * | p_motor_120_driver_instance | Motor 120 driver access module. |
| timer_instance_t const * | p_speed_cyclic_timer_instance | Cyclic process of speed control timer module. |
| timer_instance_t const * | p_speed_calc_timer_instance | Speed calculate timer module. |

### ◆ motor_120_control_sensorless_instance_ctrl_t

| struct motor_120_control_sensorless_instance_ctrl_t | | |
|---|---|---|
| 120 control sensorless instance control block | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| motor_120_control_status_t | active | Flag to set active/inactive the motor 120 control. |
| motor_120_control_run_mode_t | run_mode | Drive mode. |
| motor_120_control_timeout_error_flag_t | timeout_error_flag | Timeout error status. |
| motor_120_control_pattern_error_flag_t | pattern_error_flag | Bemf pattern error status. |
| motor_120_control_rotation_direction_t | direction | Rotational direction (0:CW ,1:CCW) |
| float | f4_ol_pattern_set_calc | Counts to change timing of open loop pattern. |
| float | f4_speed_calc_base | Base counts to calculate |

| | | |
|---|---|---|
| | | rotation speed. |
| float | f_rpm2rad | Translate value to radian/second to rpm. |
| float | f4_v_ref | Voltage reference (output of speed PI control) |
| uint32_t | u4_pwm_duty | PWM duty. |
| float | f4_ref_speed_rad | Motor speed reference. |
| float | f4_ref_speed_rad_ctrl | Motor speed reference for speed PI control. |
| float | f4_speed_rad | Motor speed. |
| uint32_t | u4_cnt_speed_pi | Counter for period of speed PI control. |
| motor_120_control_wait_stop_flag_t | flag_wait_stop | Flag for waiting for motor stop. |
| float | f4_vn_ad | Neutral voltage. |
| uint32_t | u4_cnt_adj_v | Voltage lamping count adjustment. |
| motor_120_control_sensorless_draw_in_position_t | flag_draw_in | Status of draw in a initial position. |
| motor_120_driver_phase_pattern_t | v_pattern | Voltage pattern. |
| uint32_t | u4_v_pattern_num | Selecting pattern number for openloop drive. |
| uint32_t | u4_bemf_signal | Pattern of BEMF. |
| uint32_t | u4_pre_bemf_signal | Previous pattern of BEMF. |
| motor_120_control_sensorless_pattern_change_flag_t | flag_pattern_change | Pattern change flag. |
| motor_120_control_speed_ref_t | flag_speed_ref | Speed reference flag. |
| motor_120_control_voltage_ref_t | flag_voltage_ref | Voltage reference flag. |
| uint32_t | u4_ol_signal | Pattern of BEMF. |
| uint32_t | u4_ol_pattern_set | Openloop frequency. |
| uint32_t | u4_cnt_ol_pattern_set | Counter for openloop pattern change. |
| uint32_t | u4_cnt_timeout | Counter for timeout error. |
| uint32_t | u4_bemf_timer_cnt | Value of timer counter. |
| uint32_t | u4_pre_bemf_timer_cnt | Previous value of timer counter. |
| int32_t | s4_timer_cnt_ave | Counts for 360 degrees. |
| | | |

| | | |
|---|---|---|
| uint32_t | u4_timer_cnt_buf[MOTOR_120_CONTROL_SENSORLESS_TIMES] | Counts for 60 degrees. |
| uint32_t | u4_timer_cnt_num | Array element number before 360 degrees. |
| uint32_t | u4_cnt_carrier | Counter for carrier interrupt. |
| uint32_t | u4_pre_cnt_carrier | Previous carrier interrupt count. |
| uint32_t | u4_angle_shift_cnt | Shift degrees count. |
| float | f4_pi_ctrl_err | PI control error. |
| float | f4_pi_ctrl_refi | PI control buffer of integral term. |
| motor_120_control_cfg_t const * | p_cfg | Pointer of configuration structure. |
| timer_direction_t | timer_direction | Hold timer direction. |
| timer_callback_args_t | timer_args | For call timer callbackSet function. |

## Enumeration Type Documentation

### ◆ motor_120_control_sensorless_draw_in_position_t

| enum motor_120_control_sensorless_draw_in_position_t | |
|---|---|
| Draw in a initial position | |
| Enumerator | |
| MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_INIT | Initial parameter. |
| MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_1ST_TIME | Draw in a initial position of the 1st time. |
| MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_2ND_TIME | Draw in a initial position of the 2nd time. |

### ◆ motor_120_control_sensorless_pattern_change_flag_t

| enum motor_120_control_sensorless_pattern_change_flag_t | |
|---|---|
| Pattern change | |
| Enumerator | |
| MOTOR_120_CONTROL_SENSORLESS_PATTERN_CHANGE_FLAG_CLEAR | Initial parameter. |
| MOTOR_120_CONTROL_SENSORLESS_PATTERN_CHANGE_FLAG_SET | Voltage pattern change. |

## Function Documentation

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_Open()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Open ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg* )

Opens and configures the motor sensorless 120 detection module. Implements motor_120_control_api_t::open.

Example:

```
/* Initializes the module. */
   err =
RM_MOTOR_120_CONTROL_SENSORLESS_Open(g_motor_120_control_sensorless0.p_ctrl,
                                     g_motor_120_control_sensorless0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor driver successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_Close()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Close ( motor_120_control_ctrl_t *const *p_ctrl*)

Disables specified motor sensorless 120 detection module. Implements motor_120_control_api_t::close.

Example:

```
/* Close the module. */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Close(g_motor_120_control_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_Run()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Run ( motor_120_control_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_control_api_t::run.

Example:

```
/* Start motor rotation */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Run(g_motor_120_control_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_Stop()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Stop ( motor_120_control_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Stop(g_motor_120_control_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_Reset()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Reset ( motor_120_control_ctrl_t *const *p_ctrl*)

Reset variables of motor sensorless 120 detection module. Implements motor_120_control_api_t::reset.

Example:

```
/* Reset the process. */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Reset(g_motor_120_control_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet ( motor_120_control_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed[rpm]. Implements motor_120_control_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet(g_motor_120_control_sensorless0.p_ctrl,


DEF_120_CONTROL_SENSORLESSHALL_TEST_OVSPD_LIM);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet ( motor_120_control_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get speed. Implements motor_120_control_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status* )

Get current. Implements motor_120_control_api_t::currentGet.

Example:

```
/* Get current */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_current_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet.

Example:

```
/* Get wait stop flag */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                  &smpl_wait_stop_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_timeout_error_flag_t *const *p_timeout_error_flag* )

Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet.

Example:

```
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                    &smpl_timeout_error_fl
ag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_pattern_error_flag_t *const *p_pattern_error_flag* )

Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet.

Example:

```
/* Get pattern error flag */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                    &smpl_pattern_error_fl
ag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_voltage_ref_t *const *p_voltage_ref* )

Get voltage ref. Implements motor_120_control_api_t::voltageRefGet.

Example:

```
/* Get voltage ref */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_voltage_ref);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg* )

Update the parameters of sensorless 120 detection module. Implements motor_120_control_api_t::parameterUpdate.

Example:

```
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate
(g_motor_120_control_sensorless0.p_ctrl,


g_motor_120_control_sensorless0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

## 5.2.11.2 120-degree conduction control with Hall sensors (rm_motor_120_control_hall)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_Open (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)<br>Opens and configures the motor hall 120 detection module. Implements motor_120_control_api_t::open. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_Close (motor_120_control_ctrl_t *const p_ctrl)<br>Disables specified motor hall 120 detection module. Implements motor_120_control_api_t::close. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_Run (motor_120_control_ctrl_t *const p_ctrl)<br>Run motor (Start motor rotation). Implements motor_120_control_api_t::run. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_Stop (motor_120_control_ctrl_t *const p_ctrl)<br>Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_Reset (motor_120_control_ctrl_t *const p_ctrl)<br>Reset variables of motor hall 120 detection module. Implements motor_120_control_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_SpeedSet (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed[rpm]. Implements motor_120_control_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_SpeedGet (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get speed. Implements motor_120_control_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_CurrentGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)<br>Get current. Implements motor_120_control_api_t::currentGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag)<br>Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag)<br>Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag) |

| | |
|---|---|
| | Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_VoltageRefGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref)<br>Get voltage ref. Implements motor_120_control_api_t::voltageRefGet. More... |
| fsp_err_t | RM_MOTOR_120_CONTROL_HALL_ParameterUpdate (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)<br>Update the parameters of hall 120 detection module. Implements motor_120_control_api_t::parameterUpdate. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Control Interface.

# Overview

The motor current is used to control the electric current of motor rotation in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

### Features

The motor 120 control hall module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling control.
- Voltage error compensation.

# Configuration

### Build Time Configurations for rm_motor_120_control_hall

The following build time configurations are defined in fsp_cfg/rm_motor_120_control_hall_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > 120-degree conduction control with Hall sensors

**(rm_motor_120_control_hall)**

This module can be added to the Stacks tab via New Stack > Motor > 120-degree conduction control with Hall sensors (rm_motor_120_control_hall).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_120_control_hall0 | Module name. |
| Conduction type | • First 60 degree PWM<br>• Complementary First 60 degree PWM | First 60 degree PWM | Select conduction type |
| Timeout counts (msec) | Must be a valid integer. | 200 | Counts to judge rotor unrotate |
| Maximum voltage (V) | Must be a valid non-negative value. | 20.0 | Maximum output voltage (V) |
| Minimum voltage (V) | Must be a valid non-negative value. | 3.0 | Minimum output voltage (V) |
| Speed PI decimation | Must be a valid integer. | 0 | Speed PI control decimation count |
| Freerun timer frequency (MHz) | Must be a valid integer. | 120 | Freerun timer frequency (MHz) |
| Speed LPF | Must be a valid non-negative value. | 1.0 | Speed LPF parameter |
| Step of speed reference change | Must be a valid non-negative value. | 0.2 | Speed ref change step |
| Start reference voltage (V) | Must be a valid non-negative value. | 5.8 | Reference voltage for boot mode (V) |
| Hall wait counts | Must be a valid integer. | 12 | Wait counts of hall interrupts to start speed calculation |
| Stop judge time | Must be a valid integer. | 1000 | Stop judge time |
| Minimum limit speed (rpm) | Must be a valid integer. | 550 | Minimum limit speed (rpm) (mechanical angle) |
| PI control KP | Must be a valid non-negative value. | 0.02 | PI control gain of proportional term |
| PI control KI | Must be a valid non-negative value. | 0.0005 | PI control gain of integral term |
| PI control limit | Must be a valid non-negative value. | 24.0 | PI control limit of integral term |

| | | | |
|---|---|---|---|
| Hall interrupt mask value | Must be a valid integer. | 15 | For limiting hall interrupt processing. Limited by the number of ADC interrupts |
| Motor Parameter | | | |
| Pole pairs | Must be a valid integer. | 2 | Pole pairs |
| Resistance (ohm) | Must be a valid non-negative value. | 6.447 | Resistance |
| Inductance of d-axis (H) | Must be a valid non-negative value. | 0.0045 | Inductance of d-axis |
| Inductance of q-axis (H) | Must be a valid non-negative value. | 0.0045 | Inductance of q-axis |
| Permanent magnetic flux (Wb) | Must be a valid non-negative value. | 0.02159 | Permanent magnetic flux |
| Motor Parameter > Rotor inertia (kgm^2) | Must be a valid non-negative value. | 1.8 | Rotor inertia |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | Callback function |
| Hall sensor port U | Manual Entry | BSP_IO_PORT_04_PIN_11 | Hall sensor port U |
| Hall sensor port V | Manual Entry | BSP_IO_PORT_04_PIN_10 | Hall sensor port V |
| Hall sensor port W | Manual Entry | BSP_IO_PORT_04_PIN_09 | Hall sensor port W |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

Depend on included ICU module.

## Usage Notes

### Limitations

- Set the period of current control with none-negative value.
- Set the reference voltage with none-negative value.

## Examples

### Basic Example

This is a basic example of minimal use of the motor 120 control hall in an application.

```
void motor_120_control_hall_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_CONTROL_HALL_Open(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedSet(g_motor_120_control_hall0.p_ctrl,
DEF_120_CONTROL_HALL_TEST_SPEED_REF);
 /* Start motor rotation */
    (void) RM_MOTOR_120_CONTROL_HALL_Run(g_motor_120_control_hall0.p_ctrl);
 /* Get current motor speed */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedGet(g_motor_120_control_hall0.p_ctrl,
&smpl_speed);
 /* Get current */
    (void) RM_MOTOR_120_CONTROL_HALL_CurrentGet(g_motor_120_control_hall0.p_ctrl,
&smpl_current_status);
 /* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_wait_stop_flag);
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_timeout_error_flag);
 /* Get pattern error flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_pattern_error_flag);
 /* Get voltage ref */
    (void) RM_MOTOR_120_CONTROL_HALL_VoltageRefGet(g_motor_120_control_hall0.p_ctrl,
&smpl_voltage_ref);
    (void)
```

```
RM_MOTOR_120_CONTROL_HALL_ParameterUpdate(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
 /* Stop motor rotation */
    (void) RM_MOTOR_120_CONTROL_HALL_Stop(g_motor_120_control_hall0.p_ctrl);
 /* Reset the process. */
    (void) RM_MOTOR_120_CONTROL_HALL_Reset(g_motor_120_control_hall0.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_120_CONTROL_HALL_Close(g_motor_120_control_hall0.p_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | motor_120_control_hall_extended_cfg_t |
| struct | motor_120_control_hall_instance_ctrl_t |

## Data Structure Documentation

### ◆ motor_120_control_hall_extended_cfg_t

| struct motor_120_control_hall_extended_cfg_t | | |
|---|---|---|
| Extended configurations for motor 120 control hall | | |
| Data Fields | | |
| bsp_io_port_pin_t | port_hall_sensor_u | Hall sensor port U. |
| bsp_io_port_pin_t | port_hall_sensor_v | Hall sensor port V. |
| bsp_io_port_pin_t | port_hall_sensor_w | Hall sensor port W. |
| float | f4_start_refv | Reference voltage for boot mode. |
| uint32_t | u4_hall_wait_cnt | Wait counts of hall interrupts for speed calculation. |
| uint32_t | u4_stop_judge_time | Stop judge time. |
| uint32_t | u4_min_speed_rpm | Minimum limit speed (rpm) (mechanical angle) |
| uint32_t | u4_hall_interrupt_mask_value | For limiting hall interrupt processing. Limited by the number of ADC interrupts. |
| motor_120_driver_instance_t const * | p_motor_120_driver_instance | Motor 120 driver access module. |
| timer_instance_t const * | p_speed_cyclic_timer_instance | Cyclic process of speed control timer module. |
| timer_instance_t const * | p_speed_calc_timer_instance | Speed calculate timer module. |

| | | |
|---|---|---|
| external_irq_instance_t const * | p_u_hall_irq_instance | U phase hall interrupt. |
| external_irq_instance_t const * | p_v_hall_irq_instance | V phase hall interrupt. |
| external_irq_instance_t const * | p_w_hall_irq_instance | W phase hall interrupt. |

### ◆ motor_120_control_hall_instance_ctrl_t

| struct motor_120_control_hall_instance_ctrl_t | | |
|---|---|---|
| 120 control hall instance control block | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| motor_120_control_status_t | active | Flag to set active/inactive the motor 120 control. |
| motor_120_control_run_mode_t | run_mode | Drive mode. |
| motor_120_control_timeout_error_flag_t | timeout_error_flag | Timeout error status. |
| motor_120_control_pattern_error_flag_t | pattern_error_flag | Hall pattern error status. |
| motor_120_control_rotation_direction_t | direction | Rotational direction (0:CW ,1:CCW) |
| float | f4_speed_calc_base | Base counts to calculate rotation speed. |
| float | f_rpm2rad | Translate value to radian/second to rpm. |
| float | f4_v_ref | Voltage reference (output of speed PI control) |
| float | f4_ref_speed_rad | Motor speed reference. |
| float | f4_ref_speed_rad_ctrl | Motor speed reference for speed PI control. |
| float | f4_speed_rad | Motor speed. |
| uint32_t | u4_cnt_speed_pi | Counter for period of speed PI control. |
| motor_120_control_wait_stop_flag_t | flag_wait_stop | Flag for waiting for motor stop. |
| uint32_t | u4_cnt_wait_stop | Counter for waiting motor stop. |
| motor_120_driver_phase_pattern_t | v_pattern | Voltage pattern. |
| motor_120_control_speed_ref_t | flag_speed_ref | Speed reference flag. |
| motor_120_control_voltage_ref_t | flag_voltage_ref | Voltage reference flag. |
| | | |

| | | |
|---|---|---|
| uint32_t | u4_cnt_timeout | Counter for timeout error. |
| uint32_t | u4_hall_timer_cnt | Value of timer counter. |
| uint32_t | u4_pre_hall_timer_cnt | Previous value of timer counter. |
| int32_t | s4_timer_cnt_ave | Counts for 360 degrees. |
| uint32_t | u4_timer_cnt_buf[MOTOR_120_CONTROL_HALL_TIMES] | Counts for 60 degrees. |
| uint32_t | u4_timer_cnt_num | Array element number before 360 degrees. |
| float | f4_pi_ctrl_err | PI control error. |
| float | f4_pi_ctrl_refi | PI control buffer of integral term. |
| uint32_t | u4_hall_intr_cnt | For start timing of speed calculation. |
| uint32_t | u4_adc_interrupt_cnt | Number of ADC interrupt processing. |
| motor_120_control_cfg_t const * | p_cfg | Pointer of configuration structure. |
| timer_direction_t | timer_direction | Hold timer direction. |
| external_irq_callback_args_t | hall_interrupt_args | For call IRQ callbackSet function. |
| timer_callback_args_t | timer_args | For call timer callbackSet function. |

### Function Documentation

### ◆ RM_MOTOR_120_CONTROL_HALL_Open()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Open ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg* )

Opens and configures the motor hall 120 detection module. Implements motor_120_control_api_t::open.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_120_CONTROL_HALL_Open(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor 120 driver successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_Close()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Close ( motor_120_control_ctrl_t *const *p_ctrl*)

Disables specified motor hall 120 detection module. Implements motor_120_control_api_t::close.

Example:

```
/* Close the module. */
   (void) RM_MOTOR_120_CONTROL_HALL_Close(g_motor_120_control_hall0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_HALL_Run()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Run ( motor_120_control_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_control_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_120_CONTROL_HALL_Run(g_motor_120_control_hall0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_HALL_Stop()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Stop ( motor_120_control_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_120_CONTROL_HALL_Stop(g_motor_120_control_hall0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_HALL_Reset()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Reset ( motor_120_control_ctrl_t *const *p_ctrl*)

Reset variables of motor hall 120 detection module. Implements motor_120_control_api_t::reset.

Example:

```
/* Reset the process. */
   (void) RM_MOTOR_120_CONTROL_HALL_Reset(g_motor_120_control_hall0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_HALL_SpeedSet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_SpeedSet ( motor_120_control_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed[rpm]. Implements motor_120_control_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_120_CONTROL_HALL_SpeedSet(g_motor_120_control_hall0.p_ctrl,
DEF_120_CONTROL_HALL_TEST_SPEED_REF);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_CONTROL_HALL_SpeedGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_SpeedGet ( motor_120_control_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get speed. Implements motor_120_control_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_120_CONTROL_HALL_SpeedGet(g_motor_120_control_hall0.p_ctrl,
&smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_CurrentGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_CurrentGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status* )

Get current. Implements motor_120_control_api_t::currentGet.

Example:

```
/* Get current */
   (void) RM_MOTOR_120_CONTROL_HALL_CurrentGet(g_motor_120_control_hall0.p_ctrl,
&smpl_current_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet.

Example:

```
/* Get wait stop flag */
   (void)
RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_wait_stop_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_timeout_error_flag_t *const *p_timeout_error_flag* )

Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet.

Example:

```
/* Get timeout error flag */
   (void)
RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_timeout_error_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_pattern_error_flag_t *const *p_pattern_error_flag* )

Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet.

Example:

```
/* Get pattern error flag */
   (void)
RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_pattern_error_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_VoltageRefGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_VoltageRefGet ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_voltage_ref_t *const *p_voltage_ref* )

Get voltage ref. Implements motor_120_control_api_t::voltageRefGet.

Example:

```
/* Get voltage ref */
   (void) RM_MOTOR_120_CONTROL_HALL_VoltageRefGet(g_motor_120_control_hall0.p_ctrl,
&smpl_voltage_ref);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_CONTROL_HALL_ParameterUpdate()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_ParameterUpdate ( motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg* )

Update the parameters of hall 120 detection module. Implements motor_120_control_api_t::parameterUpdate.

Example:

```
    (void)
RM_MOTOR_120_CONTROL_HALL_ParameterUpdate(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

## 5.2.11.3 ADC and PWM Modulation (rm_motor_driver)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_DRIVER_Open (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)<br>Opens and configures the Motor Driver module. Implements motor_driver_api_t::open. More... |
| fsp_err_t | RM_MOTOR_DRIVER_Close (motor_driver_ctrl_t *const p_ctrl)<br>Disables specified Motor Driver Module. Implements motor_driver_api_t::close. More... |
| fsp_err_t | RM_MOTOR_DRIVER_Reset (motor_driver_ctrl_t *const p_ctrl)<br>Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_DRIVER_PhaseVoltageSet (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage) |

| | |
|---|---|
| | Set Phase Voltage Data to calculate PWM duty. Implements motor_driver_api_t::phaseVoltageSet. More... |
| fsp_err_t | RM_MOTOR_DRIVER_CurrentGet (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get)<br>Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet. More... |
| fsp_err_t | RM_MOTOR_DRIVER_FlagCurrentOffsetGet (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)<br>Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet. More... |
| fsp_err_t | RM_MOTOR_DRIVER_CurrentOffsetRestart (motor_driver_ctrl_t *const p_ctrl)<br>Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart. More... |
| fsp_err_t | RM_MOTOR_DRIVER_ParameterUpdate (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)<br>Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor driver Interface.

# Overview

The motor driver module is used to translate phase voltage to PWM duty and output PWM, and detect phase current and main line voltage. This module should be called cyclically at included A/D Conversion finish interrupt.

### Features

The Motor Driver Module has below features.

- Calculate each phase(U/V/W) PWM duty according to reference and output PWM.
- Detect each phase current and main line voltage.
- Detect and correct A/D offset at phase current channel

# Configuration

### Build Time Configurations for rm_motor_driver

The following build time configurations are defined in fsp_cfg/rm_motor_driver_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| ADC_B Support | • Enabled<br>• Disabled | Disabled | Select ADC_B module support. |
| Shared ADC support | • Enabled<br>• Disabled | Disabled | Select Shared ADC support. |
| Supported Motor Number | Must be greater than 1. | 1 | |

### Configurations for Motor > ADC and PWM Modulation (rm_motor_driver)

This module can be added to the Stacks tab via New Stack > Motor > ADC and PWM Modulation (rm_motor_driver).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_driver0 | Module name. |
| Shunt type | • 1 shunt<br>• 2 shunt<br>• 3 shunt | 2 shunt | Select shunt type |
| Modulation method | • SVPWM<br>• SPWM | SVPWM | Select PWM modulation method |
| PWM output port UP | Manual Entry | 0 | PWM output port UP |
| PWM output port UN | Manual Entry | 0 | PWM output port UN |
| PWM output port VP | Manual Entry | 0 | PWM output port VP |
| PWM output port VN | Manual Entry | 0 | PWM output port VN |
| PWM output port WP | Manual Entry | 0 | PWM output port WP |
| PWM output port WN | Manual Entry | 0 | PWM output port WN |
| PWM Timer Frequency (MHz) | Must be a valid integer. | 120 | GPT PWM timer frequency |
| PWM Carrier Period (Microseconds) | Must be a valid integer. | 50 | GPT PWM carrier period |
| Dead Time (Raw Counts) | Must be a valid integer. | 240 | GPT PWM dead time |
| Current Range (A) | Must be a valid value | 27.5F | Current range to measure(Maximum |

| | | | |
|---|---|---|---|
| | | | input current) |
| Voltage Range (V) | Must be a valid value | 111.0F | Voltage range to measure(Maximum input Main Line Voltage) |
| Counts for current offset measurement | Must be a valid integer. | 500 | How many times to measure current offset |
| A/D conversion channel for U Phase current | Value must be a supported channel number | 0 | Specify the A/D channel for U phase current (channel availability varies by MCU) |
| A/D conversion channel for W Phase current | Value must be a supported channel number | 2 | Specify the A/D channel for W phase current (channel availability varies by MCU) |
| A/D conversion channel for Main Line Voltage | Value must be a supported channel number | 5 | Specify the A/D channel for main line voltage (channel availability varies by MCU) |
| A/D conversion channel for V Phase current | Value must be a supported channel number | 1 | Specify the A/D channel for V phase current (channel availability varies by MCU) |
| A/D conversion channel for sin signal | Value must be a supported channel number | 27 | Specify the A/D channel for sin signal of induction sensor (channel availability varies by MCU) |
| A/D conversion channel for cos signal | Value must be a supported channel number | 28 | Specify the A/D channel for cos signal of induction sensor (channel availability varies by MCU) |
| Using ADC Scan Group | Value should be between 0 to 8 (valid only with ADC_B module) | 0 | For MCUs with ADC_B, select the scan group used. |
| A/D conversion unit for U Phase current | Must be a valid non-negative value. | 0 | Select the A/D conversion module for U phase current (only valid with adc module) |
| A/D conversion unit for W Phase current | Must be a valid non-negative value. | 0 | Select the A/D conversion module for W phase current (only |

| | | | |
|---|---|---|---|
| | | | valid with adc module) |
| A/D conversion unit for main line voltage | Must be a valid non-negative value. | 0 | Select the A/D conversion module for main line voltage (only valid with adc module) |
| A/D conversion unit for V Phase current | Must be a valid non-negative value. | 0 | Select the A/D conversion module for V phase current (only valid with adc module) |
| A/D conversion unit for sin signal | Must be a valid non-negative value. | 0 | Select the A/D conversion module for sin signal of induction sensor (only valid with adc module) |
| A/D conversion unit for cos signal | Must be a valid non-negative value. | 0 | Select the A/D conversion module for cos signal of induction sensor (only valid with adc module) |
| ADC interrupt module | • 1st<br>• 2nd | 1st | Select from which module ADC intterupt happens (only valid with adc module) |
| Adjustment value to current A/D | Must be a valid non-negative value. | 20.0 | Value to adjust 1shunt A/D double buffer |
| Minimum difference of PWM duty | Must be a valid non-negative value. | 300 | Minimum difference of PWM duty |
| Adjustment delay of A/D conversion | Must be a valid non-negative value. | 240 | Adjustment delay of A/D conversion |
| 1shunt interrupt phase | • PHASE_U<br>• PHASE_V<br>• PHASE_W | PHASE_U | Select the phase which occures ADC end interrupt. (Only valid with 1shunt) |
| Input Voltage (V) | Must be a valid value | 24.0F | Input voltage |
| Resolution of A/D conversion | Must be a valid Resolution of ADC. | 0xFFF | Resolution of A/D conversion |
| Offset of A/D conversion for current | Must be a valid non-negative value. | 0x745 | Offset of A/D conversion for current |
| Conversion level of A/D conversion for voltage | Must be a valid value | 0.66F | Conversion level of A/D conversion for voltage |
| GTIOCA Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level High | Select the behavior of the output pin when the timer is stopped. |
| GTIOCB Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level High | Select the behavior of the output pin when |

| | | | |
|---|---|---|---|
| | | | the timer is stopped. |
| Modulation | | | |
| Maximum Duty | Must be a valid value | 0.9375F | Maximum duty of PWM |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt. |

### Clock Configuration

Set used clock with included GPT timer.

### Pin Configuration

Depend on included GPT Three Phase Module and ADC Module.

## Usage Notes

When shared ADC instance is used, please perform below sequence.

1. Add (New) shared ADC instance.
2. Set configurations of below ADC module for your use.
3. Select "Common | Shared ADC support" to "Enabled".

### Limitations

Basically no limitation exists.

## Examples

### Basic Example

This is a basic example of minimal use of the Motor Driver in an application.

```
void motor_driver_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_DRIVER_Open(&g_motor_driver0.p_ctrl, &g_motor_driver0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at cyclic interrupt (e.g. included GPT PWM Carrier
intterupt).
  * This implementation is an example. */
```

```
 // while (true)
    {
 /* Application work here. */
 /* Get electric current, main line voltage and maximum voltage component */
       (void) RM_MOTOR_DRIVER_CurrentGet(&g_motor_driver0.p_ctrl, &f_get_iu,
&f_get_iw, &f_get_vdc, &f_get_va_max);
 /* Get the flag of A/D convderted current offset */
       (void) RM_MOTOR_DRIVER_FlagCurrentOffsetGet(&g_motor_driver0.p_ctrl,
&u1_get_flg_offset);
 // Perform current control process here
 /* Set phase voltage */
       (void) RM_MOTOR_DRIVER_PhaseVoltageSet(&g_motor_driver0.p_ctrl, 1.0F, 1.0F,
1.0F);
       (void) RM_MOTOR_DRIVER_ParameterUpdate(&g_motor_driver0.p_ctrl,
&g_motor_driver0.p_cfg);
    }
    (void) RM_MOTOR_DRIVER_Reset(&g_motor_driver0.p_ctrl);
 //


    (void) RM_MOTOR_DRIVER_Close(&g_motor_driver0.p_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | motor_driver_shared_instance_ctrl_t |
| struct | motor_driver_extended_shared_cfg_t |

## Enumerations

| | |
|---|---|
| enum | motor_driver_select_adc_instance_t |
| enum | motor_driver_modulation_method_t |

## Data Structure Documentation

### ◆ motor_driver_shared_instance_ctrl_t

| struct motor_driver_shared_instance_ctrl_t |
|---|
| For multiple motor |
| |

| Data Fields | | |
|---|---|---|
| uint32_t | open | |
| uint8_t | registered_motor_count | Registered motor counts. |
| void * | p_context[MOTOR_DRIVER_CFG_SUPPORT_MOTOR_NUM] | |

### ◆ motor_driver_extended_shared_cfg_t

| struct motor_driver_extended_shared_cfg_t | | |
|---|---|---|
| For multiple motor | | |
| Data Fields | | |
| adc_instance_t const * | p_adc_instance_first | first ADC instance |
| adc_instance_t const * | p_adc_instance_second | second ADC instance |
| motor_driver_shared_instance_ctrl_t *const | p_shared_instance_ctrl | |

## Enumeration Type Documentation

### ◆ motor_driver_select_adc_instance_t

| enum motor_driver_select_adc_instance_t | |
|---|---|
| Support two ADC instance valid for adc | |
| Enumerator | |
| MOTOR_DRIVER_SELECT_ADC_INSTANCE_FIRST | Use first ADC instance. |
| MOTOR_DRIVER_SELECT_ADC_INSTANCE_SECOND | Use second ADC instance. |

### ◆ motor_driver_modulation_method_t

| enum motor_driver_modulation_method_t | |
|---|---|
| Enumerator | |
| MOTOR_DRIVER_MODULATION_METHOD_SPWM | Sinusoidal pulse-width-modulation. |
| MOTOR_DRIVER_MODULATION_METHOD_SVPWM | Space vector pulse-width-modulation. |

## Function Documentation

### ◆ RM_MOTOR_DRIVER_Open()

fsp_err_t RM_MOTOR_DRIVER_Open ( motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg* )

Opens and configures the Motor Driver module. Implements motor_driver_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor Driver successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ RM_MOTOR_DRIVER_Close()

fsp_err_t RM_MOTOR_DRIVER_Close ( motor_driver_ctrl_t *const *p_ctrl*)

Disables specified Motor Driver Module. Implements motor_driver_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_DRIVER_Reset()

fsp_err_t RM_MOTOR_DRIVER_Reset ( motor_driver_ctrl_t *const *p_ctrl*)

Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_DRIVER_PhaseVoltageSet()

fsp_err_t RM_MOTOR_DRIVER_PhaseVoltageSet ( motor_driver_ctrl_t *const *p_ctrl*, float const *u_voltage*, float const *v_voltage*, float const *w_voltage* )

Set Phase Voltage Data to calculate PWM duty. Implements motor_driver_api_t::phaseVoltageSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_DRIVER_CurrentGet()

fsp_err_t RM_MOTOR_DRIVER_CurrentGet ( motor_driver_ctrl_t *const *p_ctrl*, motor_driver_current_get_t *const *p_current_get* )

Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_DRIVER_FlagCurrentOffsetGet()

fsp_err_t RM_MOTOR_DRIVER_FlagCurrentOffsetGet ( motor_driver_ctrl_t *const *p_ctrl*, uint8_t *const *p_flag_offset* )

Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet .

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_DRIVER_CurrentOffsetRestart()

fsp_err_t RM_MOTOR_DRIVER_CurrentOffsetRestart ( motor_driver_ctrl_t *const *p_ctrl*)

Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully restarted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_DRIVER_ParameterUpdate()

fsp_err_t RM_MOTOR_DRIVER_ParameterUpdate ( motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg* )

Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## 5.2.11.4 ADC and PWM modulation (rm_motor_120_driver)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_DRIVER_Open (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)<br>Opens and configures the motor 120 driver module. Implements motor_120_driver_api_t::open. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_Close (motor_120_driver_ctrl_t *const p_ctrl)<br>Disables specified motor 120 driver module. Implements motor_120_driver_api_t::close. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_Run (motor_120_driver_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | Run motor (Start motor rotation). Implements motor_120_driver_api_t::run. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_Stop (motor_120_driver_ctrl_t *const p_ctrl)<br>Stop motor (Stop motor rotation). Implements motor_120_driver_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_Reset (motor_120_driver_ctrl_t *const p_ctrl)<br>Reset variables of motor 120 driver module. Implements motor_120_driver_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_PhaseVoltageSet (motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)<br>Set phase voltage data to calculate PWM duty. Implements motor_120_driver_api_t::phaseVoltageSet. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_PhasePatternSet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern)<br>Set phase voltage pattern. Implements motor_120_driver_api_t::phasePatternSet. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_CurrentGet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)<br>Get calculated phase current, Vdc & Va_max data. Implements motor_120_driver_api_t::currentGet. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_CurrentOffsetCalc (motor_120_driver_ctrl_t *const p_ctrl)<br>current offset detection. Implements motor_120_driver_api_t::currentOffsetCalc More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset)<br>Get the flag of finish current offset detection. Implements motor_120_driver_api_t::flagCurrentOffsetGet. More... |
| fsp_err_t | RM_MOTOR_120_DRIVER_ParameterUpdate (motor_120_driver_ctrl_t |

*const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)

Update the parameters of 120 driver module. Implements motor_120_driver_api_t::parameterUpdate. More...

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor 120-Degree Driver Interface.

# Overview

The motor 120 degree driver module is used to translate phase voltage to PWM duty and output PWM, and detect phase current, voltage and main line voltage. This module should be called cyclically at included A/D conversion finish interrupt.

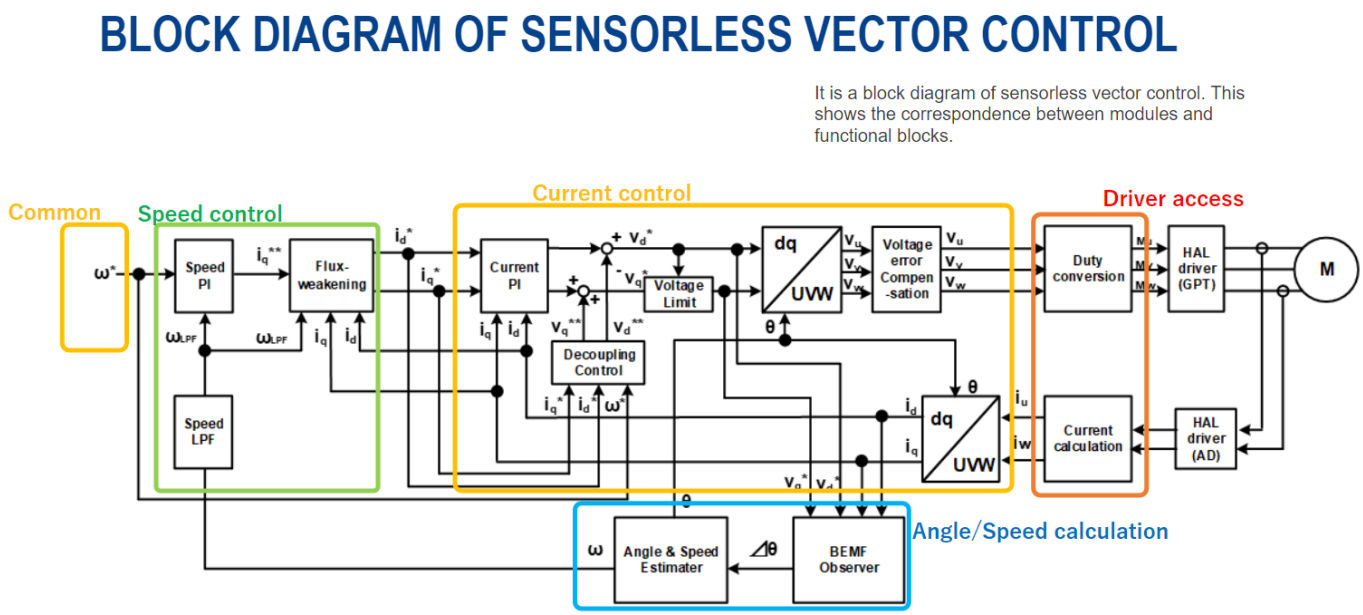



Figure 261: Image of Current Control Module(yellow block)

### Features

The motor 120 degree driver module has below features.

- Calculate each phase(U/V/W) PWM duty according to reference and output PWM.
- Detect each phase current, phase voltage and main line voltage.
- Detect and correct A/D offset at phase current and voltage channel

# Configuration

### Build Time Configurations for rm_motor_120_driver

The following build time configurations are defined in fsp_cfg/rm_motor_120_driver_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| ADC_B Support | • Enabled<br>• Disabled | Disabled | Select ADC_B module support. |
| Shared ADC support | • Enabled<br>• Disabled | Disabled | Select Shared ADC support. |
| Support Motor Number | Must be greater than 1. | 1 | |

**Configurations for Motor > ADC and PWM modulation (rm_motor_120_driver)**

This module can be added to the Stacks tab via New Stack > Motor > ADC and PWM modulation (rm_motor_120_driver).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_120_driver0 | Module name. |
| 120 degree control type | • Sensorless<br>• Hall | Sensorless | 120 degree control type |
| PWM output port UP | Manual Entry | BSP_IO_PORT_06_PIN_01 | PWM output port UP |
| PWM output port UN | Manual Entry | BSP_IO_PORT_06_PIN_00 | PWM output port UN |
| PWM output port VP | Manual Entry | BSP_IO_PORT_01_PIN_13 | PWM output port VP |
| PWM output port VN | Manual Entry | BSP_IO_PORT_01_PIN_14 | PWM output port VN |
| PWM output port WP | Manual Entry | BSP_IO_PORT_01_PIN_11 | PWM output port WP |
| PWM output port WN | Manual Entry | BSP_IO_PORT_01_PIN_12 | PWM output port WN |
| PWM timer frequency (MHz) | Must be a valid positive integer. | 120 | GPT PWM timer frequency |
| PWM carrier period (Microseconds) | Must be a valid positive integer. | 50 | GPT PWM carrier period |
| Dead time (Raw counts) | Must be a valid positive integer. | 240 | GPT PWM dead time |
| Current range (A) | Must be a valid non-negative value. | 27.5 | Current range to measure(Maximum input current) |
| Voltage range (V) | Must be a valid non- | 111.0 | Voltage range to |

| | | | |
|---|---|---|---|
| | negative value. | | measure(Maximum input main line voltage) |
| Resolution of A/D conversion | Must be a valid Resolution of ADC. | 0xFFF | Resolution of A/D conversion |
| Offset of A/D conversion for current | Must be a valid non-negative value. | 0x745 | Offset of A/D conversion for current |
| Conversion level of A/D conversion for voltage | Must be a valid non-negative value. | 0.66 | Conversion level of A/D conversion for voltage |
| Counts for current offset measurement | Must be a valid positive integer. | 500 | How many times to measure current offset |
| Input voltage | Must be a valid non-negative value. | 24.0 | Input voltage |
| A/D conversion channel for U phase current | Value must be a supported channel number | 0 | Specify the A/D channel for U phase current |
| A/D conversion channel for W phase current | Value must be a supported channel number | 2 | Specify the A/D channel for W phase current |
| A/D conversion channel for main line voltage | Value must be a supported channel number | 5 | Specify the A/D channel for main line voltage |
| A/D conversion channel for U phase voltage | Value must be a supported channel number | 18 | Specify the A/D channel for U phase voltage |
| A/D conversion channel for V phase voltage | Value must be a supported channel number | 20 | Specify the A/D channel for V phase voltage |
| A/D conversion channel for W phase voltage | Value must be a supported channel number | 6 | Specify the A/D channel for W phase voltage |
| A/D conversion unit for U phase current | Must be a valid non-negative value. | 0 | Specify the A/D unit for U phase current |
| A/D conversion unit for W phase current | Must be a valid non-negative value. | 0 | Specify the A/D unit for W phase current |
| A/D conversion unit for main line voltage | Must be a valid non-negative value. | 0 | Specify the A/D unit for main line voltage |
| A/D conversion unit for U phase voltage | Must be a valid non-negative value. | 0 | Specify the A/D unit for U phase voltage |
| A/D conversion unit for V phase voltage | Must be a valid non-negative value. | 0 | Specify the A/D unit for V phase voltage |
| A/D conversion unit for W phase voltage | Must be a valid non-negative value. | 0 | Specify the A/D unit for W phase voltage |
| GTIOCA stop level | • Pin Level Low | Pin Level High | Select the behavior of |

| | | | |
|---|---|---|---|
| | • Pin Level High | | the output pin when the timer is stopped. |
| GTIOCB stop level | • Pin Level Low<br>• Pin Level High | Pin Level High | Select the behavior of the output pin when the timer is stopped. |
| ADC interrupt module | • 1st<br>• 2nd | 1st | Select from which module ADC intterupt happens |
| Modulation | | | |
| Maximum duty | Must be a valid non-negative value. | 0.9375 | Maximum duty of PWM |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt. |

### Clock Configuration

Set used clock with included GPT timer.

### Pin Configuration

Depend on included GPT three phase module and ADC module.

## Usage Notes

### Limitations

Basically no limitation exists.

## Examples

### Basic Example

This is a basic example of minimal use of the Motor 120 degree driver in an application.

```
void motor_120_driver_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_DRIVER_Open(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
 /* Start PWM output */
```

```
    err = RM_MOTOR_120_DRIVER_Run(&g_motor_120_driver0.p_ctrl);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at cyclic interrupt (e.g. included GPT PWM carrier
intterupt).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Caclutarion of current offset */
       (void) RM_MOTOR_120_DRIVER_CurrentOffsetCalc(&g_motor_120_driver0.p_ctrl);
 /* Get electric current, main line voltage and maximum voltage component */
       (void) RM_MOTOR_120_DRIVER_CurrentGet(&g_motor_120_driver0.p_ctrl,
&g_current_status);
 /* Get the flag of A/D convderted current offset */
       (void) RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet(&g_motor_120_driver0.p_ctrl,
&u1_get_flg_offset);
 // Perform current control process here
 /* Set phase voltage */
       (void) RM_MOTOR_120_DRIVER_PhaseVoltageSet(&g_motor_120_driver0.p_ctrl, 1.0F,
1.0F, 1.0F);
 /* Set phase pattern */
       (void) RM_MOTOR_120_DRIVER_PhasePatternSet(&g_motor_120_driver0.p_ctrl,
MOTOR_120_DRIVER_API_VP_ON_WN_PWM);
       (void) RM_MOTOR_120_DRIVER_ParameterUpdate(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
    }
    (void) RM_MOTOR_120_DRIVER_Stop(&g_motor_120_driver0.p_ctrl);
    (void) RM_MOTOR_120_DRIVER_Reset(&g_motor_120_driver0.p_ctrl);
 //


    (void) RM_MOTOR_120_DRIVER_Close(&g_motor_120_driver0.p_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | motor_120_driver_shared_instance_ctrl_t |

| | |
|---|---|
| struct | motor_120_driver_modulation_t |
| struct | motor_120_driver_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | motor_120_driver_select_adc_instance_t |
| enum | motor_120_driver_status_t |
| enum | motor_120_driver_type_t |

## Data Structure Documentation

### ◆ motor_120_driver_shared_instance_ctrl_t

| struct motor_120_driver_shared_instance_ctrl_t | | |
|---|---|---|
| For multiple ADC module | | |
| Data Fields | | |
| uint32_t | open | |
| uint8_t | registered_motor_count | Registered motor counts. |
| void * | p_context[MOTOR_120_DRIVER_CFG_SUPPORT_MOTOR_NUM] | |

### ◆ motor_120_driver_modulation_t

| struct motor_120_driver_modulation_t | | |
|---|---|---|
| Modulation parameter | | |
| Data Fields | | |
| float | f4_vdc | Main line voltage (Vdc) (V) |
| float | f4_max_duty | Maximum duty cycle. |
| float | f4_min_duty | Minimum duty cycle. |
| float | f4_neutral_duty | Duty cycle that represents 0 (V) |

### ◆ motor_120_driver_extended_cfg_t

| struct motor_120_driver_extended_cfg_t | | |
|---|---|---|
| Extended configurations for motor 120 driver | | |
| Data Fields | | |
| adc_instance_t const * | p_adc_instance | ADC module instance. |
| three_phase_instance_t const * | p_three_phase_instance | PWM output module instance (GPT three phase) |
| motor_120_driver_type_t | motor_120_type | 120 degree control type |

| | | |
|---|---|---|
| adc_channel_t | iu_ad_ch | A/D channel for U phase current. |
| adc_channel_t | iw_ad_ch | A/D channel for W phase current. |
| adc_channel_t | vdc_ad_ch | A/D channel for main line voltage. |
| adc_channel_t | vu_ad_ch | A/D channel for U phase voltage. |
| adc_channel_t | vv_ad_ch | A/D channel for V phase voltage. |
| adc_channel_t | vw_ad_ch | A/D channel for W phase voltage. |
| uint8_t | iu_ad_unit | Used A/D unit number for U phase current. |
| uint8_t | iw_ad_unit | Used A/D unit number for W phase current. |
| uint8_t | vdc_ad_unit | Used A/D unit number for main line voltage. |
| uint8_t | vu_ad_unit | Used A/D unit number for U phase voltage. |
| uint8_t | vv_ad_unit | Used A/D unit number for V phase voltage. |
| uint8_t | vw_ad_unit | Used A/D unit number for W phase voltage. |
| bsp_io_port_pin_t | port_up | PWM output port UP. |
| bsp_io_port_pin_t | port_un | PWM output port UN. |
| bsp_io_port_pin_t | port_vp | PWM output port VP. |
| bsp_io_port_pin_t | port_vn | PWM output port VN. |
| bsp_io_port_pin_t | port_wp | PWM output port WP. |
| bsp_io_port_pin_t | port_wn | PWM output port WN. |
| uint32_t | u4_pwm_timer_freq | PWM timer frequency (MHz) |
| float | pwm_carrier_freq | PWM carrier frequency (kHz) |
| uint32_t | u4_deadtime | PWM deadtime (usec) |
| float | f_current_range | A/D current measure range (max current) (A) |
| float | f_vdc_range | A/D main line voltage measure range (max voltage) (V) |
| float | f_ad_resolution | A/D resolution. |
| float | f_ad_current_offset | A/D offset (Center value) |

| | | |
|---|---|---|
| float | f_ad_voltage_conversion | A/D conversion level. |
| uint32_t | u4_offset_calc_count | Calculation counts for current offset. |
| motor_120_driver_modulation_t | mod_param | Modulation parameter. |
| motor_120_driver_select_adc_instance_t | interrupt_adc | Select which interrupt to use. |
| motor_120_driver_extended_shared_cfg_t const * | p_shared_cfg | shared extended config |

## Enumeration Type Documentation

### ◆ motor_120_driver_select_adc_instance_t

| enum motor_120_driver_select_adc_instance_t | |
|---|---|
| Support two ADC instance valid for adc | |
| Enumerator | |
| MOTOR_120_DRIVER_SELECT_ADC_INSTANCE_1ST | Use 1st ADC instance. |
| MOTOR_120_DRIVER_SELECT_ADC_INSTANCE_2ND | Use 2nd ADC instance. |

### ◆ motor_120_driver_status_t

| enum motor_120_driver_status_t | |
|---|---|
| 120 driver active flag | |
| Enumerator | |
| MOTOR_120_DRIVER_STATUS_INACTIVE | 120 driver status inactive |
| MOTOR_120_DRIVER_STATUS_ACTIVE | 120 driver status active |

### ◆ motor_120_driver_type_t

| enum motor_120_driver_type_t | |
|---|---|
| 120 degree control type | |
| Enumerator | |
| MOTOR_120_DRIVER_TYPE_SENSORLESS | 120 degree sensorless control |
| MOTOR_120_DRIVER_TYPE_HALL | 120 degree hall control |

## Function Documentation

### ◆ RM_MOTOR_120_DRIVER_Open()

fsp_err_t RM_MOTOR_120_DRIVER_Open ( motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_cfg_t const *const *p_cfg* )

Opens and configures the motor 120 driver module. Implements motor_120_driver_api_t::open.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_120_DRIVER_Open(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor 120 driver successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_DRIVER_Close()

fsp_err_t RM_MOTOR_120_DRIVER_Close ( motor_120_driver_ctrl_t *const *p_ctrl*)

Disables specified motor 120 driver module. Implements motor_120_driver_api_t::close.

Example:

```
   (void) RM_MOTOR_120_DRIVER_Close(&g_motor_120_driver0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_Run()

fsp_err_t RM_MOTOR_120_DRIVER_Run ( motor_120_driver_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_driver_api_t::run.

Example:

```
/* Start PWM output */
   err = RM_MOTOR_120_DRIVER_Run(&g_motor_120_driver0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_Stop()

fsp_err_t RM_MOTOR_120_DRIVER_Stop ( motor_120_driver_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_driver_api_t::stop.

Example:

```
   (void) RM_MOTOR_120_DRIVER_Stop(&g_motor_120_driver0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_Reset()

fsp_err_t RM_MOTOR_120_DRIVER_Reset ( motor_120_driver_ctrl_t *const *p_ctrl*)

Reset variables of motor 120 driver module. Implements motor_120_driver_api_t::reset.

Example:

```
   (void) RM_MOTOR_120_DRIVER_Reset(&g_motor_120_driver0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_PhaseVoltageSet()

fsp_err_t RM_MOTOR_120_DRIVER_PhaseVoltageSet ( motor_120_driver_ctrl_t *const *p_ctrl*, float const *u_voltage*, float const *v_voltage*, float const *w_voltage* )

Set phase voltage data to calculate PWM duty. Implements motor_120_driver_api_t::phaseVoltageSet.

Example:

```
 /* Set phase voltage */
      (void) RM_MOTOR_120_DRIVER_PhaseVoltageSet(&g_motor_120_driver0.p_ctrl, 1.0F,
1.0F, 1.0F);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_PhasePatternSet()

fsp_err_t RM_MOTOR_120_DRIVER_PhasePatternSet ( motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_phase_pattern_t const *pattern* )

Set phase voltage pattern. Implements motor_120_driver_api_t::phasePatternSet.

Example:

```
/* Set phase pattern */
      (void) RM_MOTOR_120_DRIVER_PhasePatternSet(&g_motor_120_driver0.p_ctrl,
MOTOR_120_DRIVER_API_VP_ON_WN_PWM);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_120_DRIVER_CurrentGet()

fsp_err_t RM_MOTOR_120_DRIVER_CurrentGet ( motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status* )

Get calculated phase current, Vdc & Va_max data. Implements motor_120_driver_api_t::currentGet .

Example:

```
/* Get electric current, main line voltage and maximum voltage component */
      (void) RM_MOTOR_120_DRIVER_CurrentGet(&g_motor_120_driver0.p_ctrl,
&g_current_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_DRIVER_CurrentOffsetCalc()

fsp_err_t RM_MOTOR_120_DRIVER_CurrentOffsetCalc ( motor_120_driver_ctrl_t *const *p_ctrl*)

current offset detection. Implements motor_120_driver_api_t::currentOffsetCalc

Example:

```
/* Caclutarion of current offset */
      (void) RM_MOTOR_120_DRIVER_CurrentOffsetCalc(&g_motor_120_driver0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet()

fsp_err_t RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet ( motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_flag_offset_calc_t *const *p_flag_offset* )

Get the flag of finish current offset detection. Implements motor_120_driver_api_t::flagCurrentOffsetGet.

Example:

```
/* Get the flag of A/D convderted current offset */
      (void) RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet(&g_motor_120_driver0.p_ctrl,
&u1_get_flg_offset);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_120_DRIVER_ParameterUpdate()

| fsp_err_t RM_MOTOR_120_DRIVER_ParameterUpdate ( motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_cfg_t const *const *p_cfg* ) |
|---|
| Update the parameters of 120 driver module. Implements motor_120_driver_api_t::parameterUpdate.<br><br>Example:<br>`(void) RM_MOTOR_120_DRIVER_ParameterUpdate(&g_motor_120_driver0.p_ctrl, &g_motor_120_driver0.p_cfg);`<br><br>**Return values**<br>FSP_SUCCESS — Successfully data was updated.<br>FSP_ERR_ASSERTION — Null pointer.<br>FSP_ERR_NOT_OPEN — Module is not open.<br>FSP_ERR_INVALID_ARGUMENT — Input parameter error. |

## 5.2.11.5 Motor 120 degree control (rm_motor_120_degree)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_DEGREE_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)<br>Configure the motor in register start mode. Implements motor_api_t::open. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_Close (motor_ctrl_t *const p_ctrl)<br>Disables specified motor control block. Implements motor_api_t::close. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_Reset (motor_ctrl_t *const p_ctrl)<br>Reset motor control block. Implements motor_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_Run (motor_ctrl_t *const p_ctrl)<br>Run motor (Start motor rotation). Implements motor_api_t::run. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_DEGREE_Stop (motor_ctrl_t *const p_ctrl)<br>Stop motor (Stop motor rotation). Implements motor_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error)<br>Set error information. Implements motor_api_t::errorSet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed reference[rpm]. Implements motor_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)<br>Get current control status. Implements motor_api_t::statusGet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get rotational speed. Implements motor_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag_wait_stop)<br>Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)<br>Check the occurunce of error. Implements motor_api_t::errorCheck. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)<br>Set position reference. Implements motor_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_120_DEGREE_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)<br>Set position reference. Implements motor_api_t::angleGet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_120_DEGREE_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)<br><br>Select using function. Implements motor_api_t::functionSelect. More... |

### Detailed Description

Usual control of a SPM (Surface Permanent Magnet) motor on RA MCUs. This module implements the Motor 120 degree control (rm_motor_120_degree).

# Overview

The motor 120 degree control is used to control a motor rotation in an application. This module is implemented with using a SPM motor. User can start/stop motor rotation simply.

### Features

The motor 120 degree module has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

### Target Hardware

The below figure shows examples of target hardware of this Motor 120-degree Module.

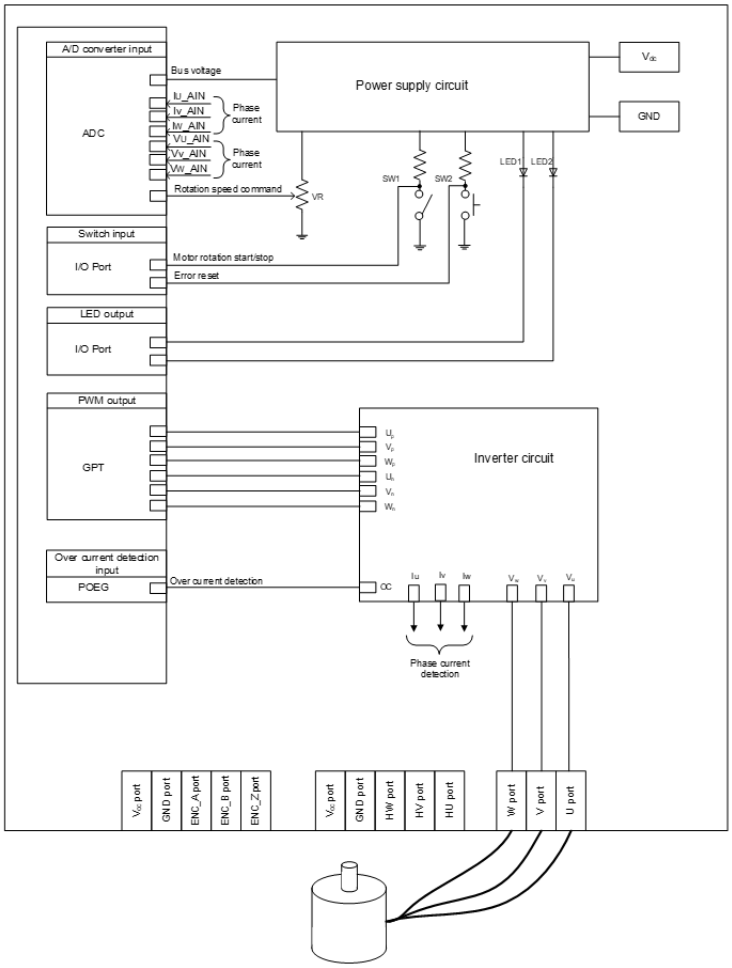



Figure 262: Example of target hardware of motor 120-degree sensorless

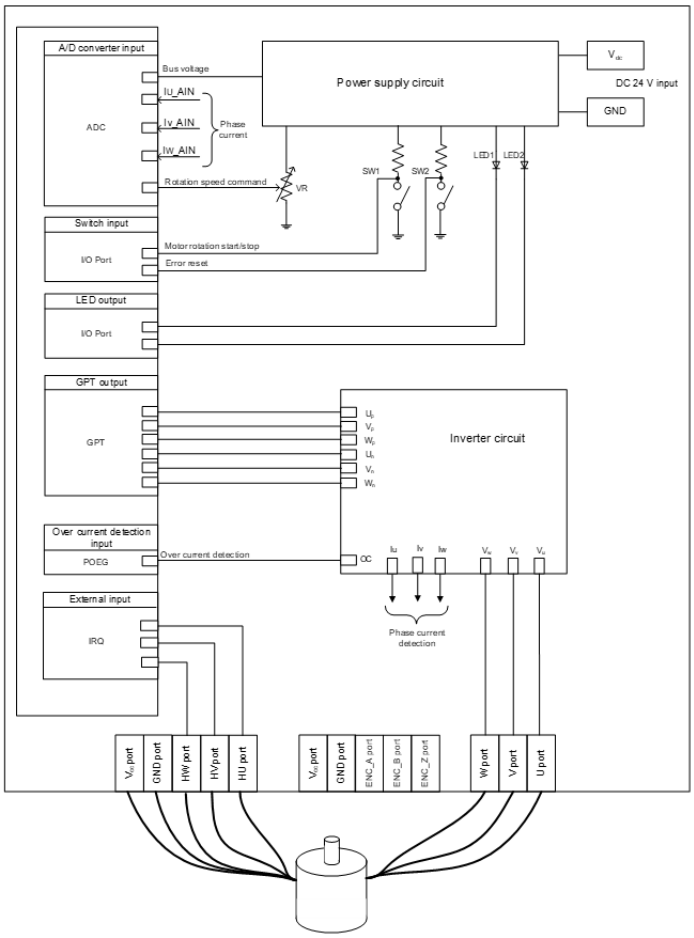



Figure 263: Example of target hardware of motor 120-degree with hall sensor

### Modulation

The modulation factor "m" is defined as follows.

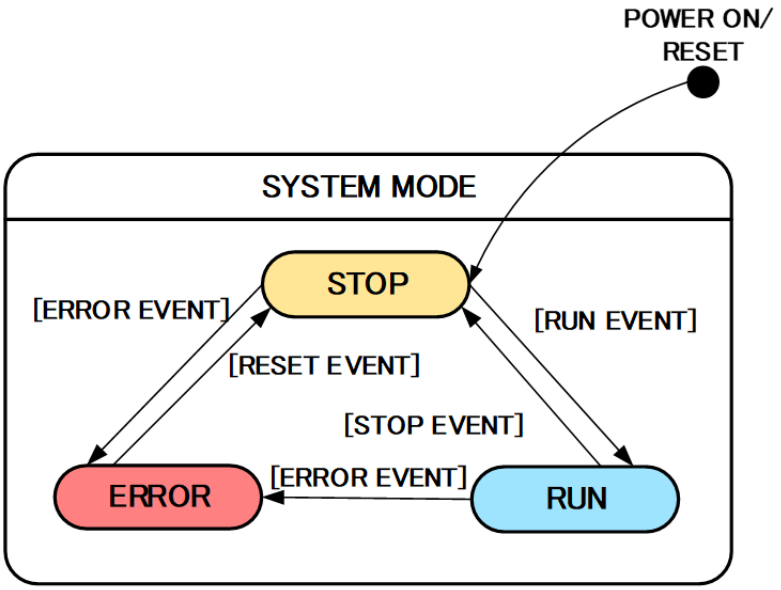
$$m = \frac{V}{E}$$

m : Modulation ratio  V : Reference voltage  E : Inverter input voltage

Figure 264: Modulation factor

### State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".




| | | MODE | | |
|---|---|---|---|---|
| | | STOP | RUN | ERROR |
| EVENT | STOP | STOP | STOP | ERROR |
| | RUN | RUN | RUN | ERROR |
| | ERROR | ERROR | ERROR | ERROR |
| | RESET | STOP | RUN | STOP |

Figure 265: State transition diagram

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

| EVENT name | Occurrence factor |
|---|---|
| STOP | by user operation |
| RUN | by user operation |
| ERROR | when the system detects an error |
| RESET | by user operation |

**Flowchart**

The below figures show flowcharts of motor 120-degree module.

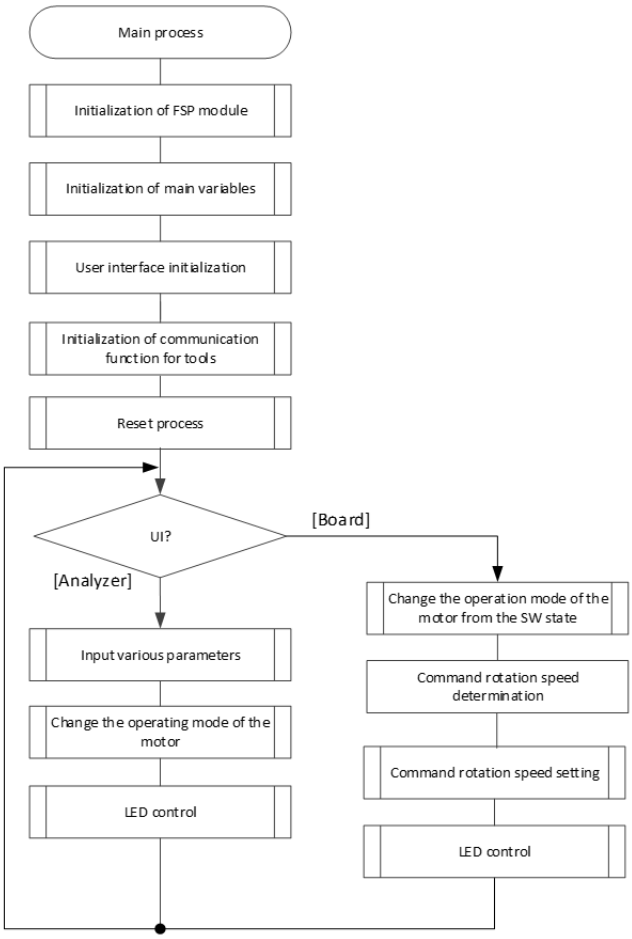



Figure 266: Main process

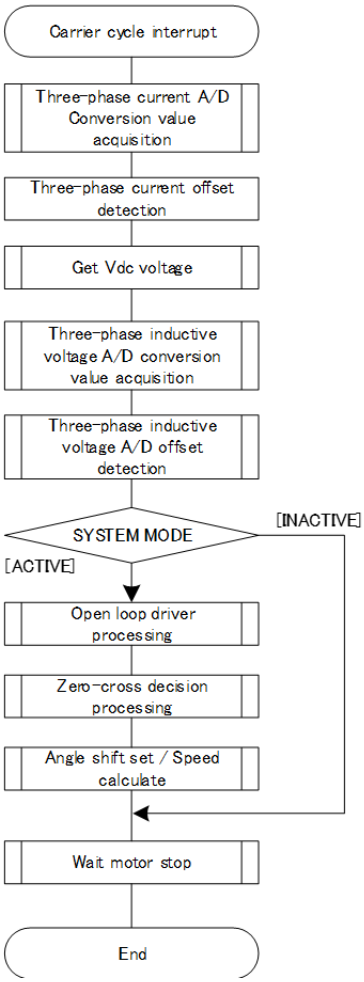



Figure 267: Current control process of 120-degree sensorless

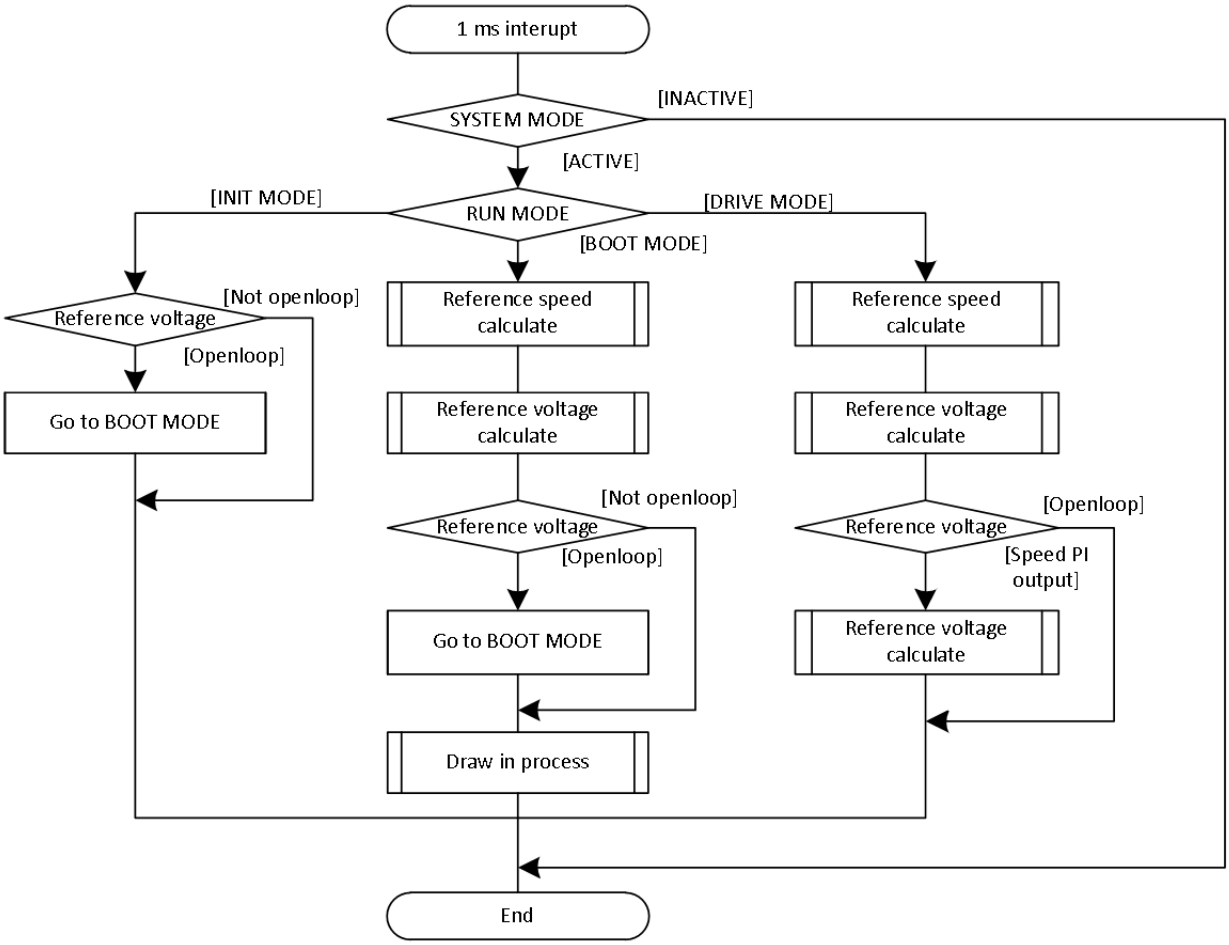



Figure 268: Speed control process of 120-degree sensorless

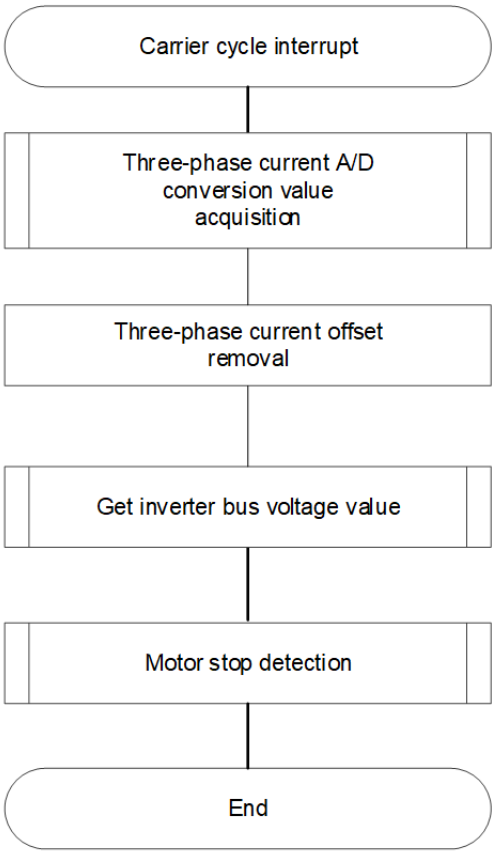



Figure 269: Current control process of 120-degree with hall sensor

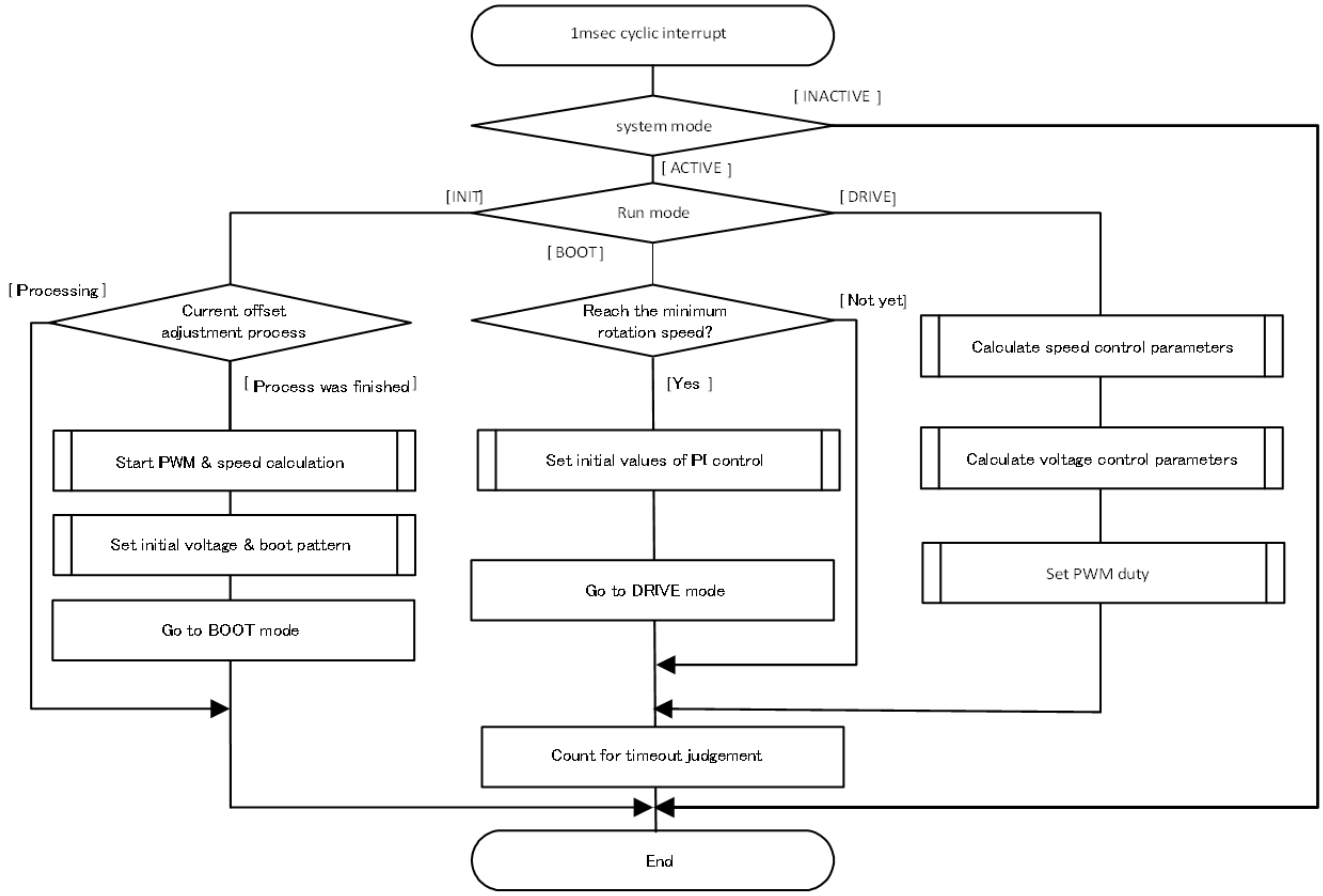



Figure 270: Speed control process of 120-degree with hall sensor

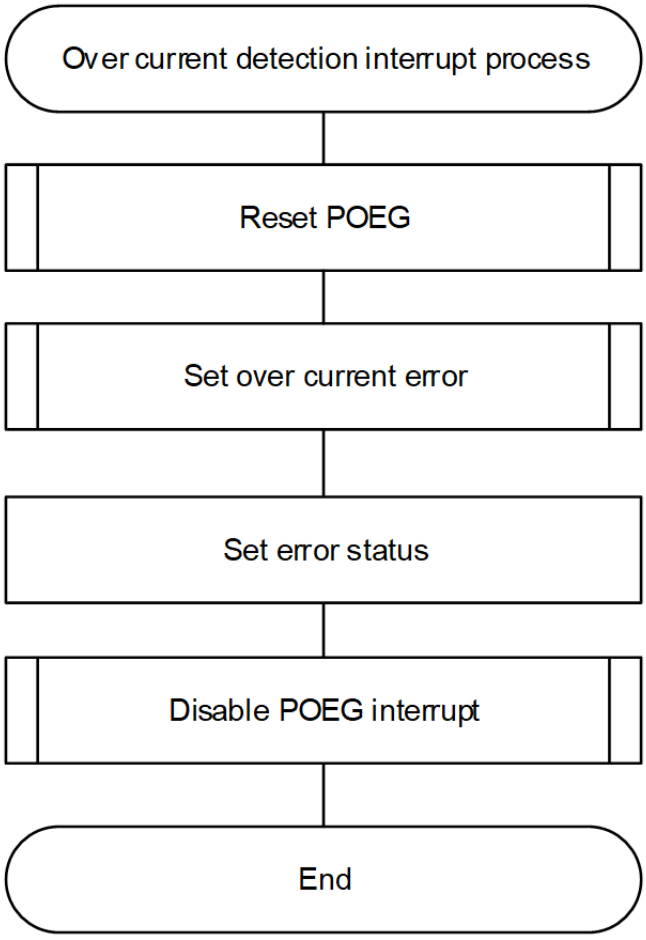



Figure 271: Over current detection interrupt process

# Configuration

### Build Time Configurations for rm_motor_120_degree

The following build time configurations are defined in fsp_cfg/rm_motor_120_degree_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor 120 degree control (rm_motor_120_degree)

This module can be added to the Stacks tab via New Stack > Motor > Motor 120 degree control (rm_motor_120_degree).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_motor_120_degree0 | Module name. |
| Limit of over current (A) | Must be a valid value | 4.0 | Limit of over current.(Detection threshold) |
| Limit of over voltage (V) | Must be a valid value | 28.0 | Limit of over voltage.(Detection threshold) |
| Limit of over speed (rpm) | Must be a valid value | 3000.0 | Limit of over speed.(Detection threshold) |
| Limit of low voltage (V) | Must be a valid value | 14.0 | Limit of low voltage.(Detection threshold) |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

### Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

# Usage Notes

### Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

# Examples

### Basic Example

This is a basic example of minimal use of the motor 120 degree in an application.

```
void motor_120_degree_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_DEGREE_Open(g_motor_120_degree0.p_ctrl,
g_motor_120_degree0.p_cfg);
```

```
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_120_DEGREE_SpeedSet(g_motor_120_degree0.p_ctrl,
DEF_120_DEGREE_SPEED_REF);
 /* Start motor rotation */
    (void) RM_MOTOR_120_DEGREE_Run(g_motor_120_degree0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_120_DEGREE_StatusGet(g_motor_120_degree0.p_ctrl, &smpl_status);
 /* Get current motor speed */
    (void) RM_MOTOR_120_DEGREE_SpeedGet(g_motor_120_degree0.p_ctrl, &smpl_speed);
 /* Get wait stop flag */
    (void) RM_MOTOR_120_DEGREE_WaitStopFlagGet(g_motor_120_degree0.p_ctrl,
&smpl_wait_stop_flag);
 /* Check error */
    (void) RM_MOTOR_120_DEGREE_ErrorCheck(g_motor_120_degree0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_120_DEGREE_Stop(g_motor_120_degree0.p_ctrl);
    (void) RM_MOTOR_120_DEGREE_ErrorSet(g_motor_120_degree0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset the process. */
    (void) RM_MOTOR_120_DEGREE_Reset(g_motor_120_degree0.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_120_DEGREE_Close(g_motor_120_degree0.p_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | motor_120_degree_statemachine_t |
| struct | motor_120_degree_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | motor_120_degree_ctrl_status_t |
| enum | motor_120_degree_ctrl_event_t |

## Data Structure Documentation

### ◆ motor_120_degree_statemachine_t

| struct motor_120_degree_statemachine_t | | |
|---|---|---|
| Statemachine structure for motor 120 degree | | |
| Data Fields | | |
| motor_120_degree_ctrl_status_t | status | The current system status. |
| motor_120_degree_ctrl_status_t | status_next | The next system status. |
| motor_120_degree_ctrl_event_t | current_event | The current event index. |
| uint16_t | u2_error_status | The error information. |

### ◆ motor_120_degree_extended_cfg_t

| struct motor_120_degree_extended_cfg_t | | |
|---|---|---|
| Extended configurations for motor 120 degree | | |
| Data Fields | | |
| motor_120_control_instance_t const * | p_motor_120_control_instance | 120 degree control Instance |
| float | f_overcurrent_limit | Over-current limit (A) |
| float | f_overvoltage_limit | Over-voltage limit (V) |
| float | f_overspeed_limit | Over-speed limit (rpm) |
| float | f_lowvoltage_limit | Low-voltage limit (V) |

## Enumeration Type Documentation

### ◆ motor_120_degree_ctrl_status_t

| enum motor_120_degree_ctrl_status_t | |
|---|---|
| Control state | |
| Enumerator | |
| MOTOR_120_DEGREE_CTRL_STATUS_STOP | Stop mode. |
| MOTOR_120_DEGREE_CTRL_STATUS_RUN | Run mode. |
| MOTOR_120_DEGREE_CTRL_STATUS_ERROR | Error mode. |

### ◆ motor_120_degree_ctrl_event_t

| enum motor_120_degree_ctrl_event_t | |
|---|---|
| Control event | |
| Enumerator | |
| MOTOR_120_DEGREE_CTRL_EVENT_STOP | Stop event. |
| MOTOR_120_DEGREE_CTRL_EVENT_RUN | Run event. |
| MOTOR_120_DEGREE_CTRL_EVENT_ERROR | Error event. |
| MOTOR_120_DEGREE_CTRL_EVENT_RESET | Reset event. |

## Function Documentation

### ◆ RM_MOTOR_120_DEGREE_Open()

fsp_err_t RM_MOTOR_120_DEGREE_Open ( motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg* )

Configure the motor in register start mode. Implements motor_api_t::open.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_120_DEGREE_Open(g_motor_120_degree0.p_ctrl,
g_motor_120_degree0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

***Note***

*This function should only be called once as motor configuration registers can only be written to once so subsequent calls will have no effect.*

### ◆ RM_MOTOR_120_DEGREE_Close()

fsp_err_t RM_MOTOR_120_DEGREE_Close ( motor_ctrl_t *const *p_ctrl*)

Disables specified motor control block. Implements motor_api_t::close.

Example:

```
/* Close the module. */
   (void) RM_MOTOR_120_DEGREE_Close(g_motor_120_degree0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_Reset()

fsp_err_t RM_MOTOR_120_DEGREE_Reset ( motor_ctrl_t *const *p_ctrl*)

Reset motor control block. Implements motor_api_t::reset.

Example:

```
/* Reset the process. */
   (void) RM_MOTOR_120_DEGREE_Reset(g_motor_120_degree0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_Run()

fsp_err_t RM_MOTOR_120_DEGREE_Run ( motor_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_120_DEGREE_Run(g_motor_120_degree0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_Stop()

fsp_err_t RM_MOTOR_120_DEGREE_Stop ( motor_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_120_DEGREE_Stop(g_motor_120_degree0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_ErrorSet()

fsp_err_t RM_MOTOR_120_DEGREE_ErrorSet ( motor_ctrl_t *const *p_ctrl*, motor_error_t const *error* )

Set error information. Implements motor_api_t::errorSet.

Example:

```
    (void) RM_MOTOR_120_DEGREE_ErrorSet(g_motor_120_degree0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set error information. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_SpeedSet()

fsp_err_t RM_MOTOR_120_DEGREE_SpeedSet ( motor_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
 /* Set speed reference before motor run */
    (void) RM_MOTOR_120_DEGREE_SpeedSet(g_motor_120_degree0.p_ctrl,
DEF_120_DEGREE_SPEED_REF);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set speed reference. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_StatusGet()

fsp_err_t RM_MOTOR_120_DEGREE_StatusGet ( motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status* )

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_120_DEGREE_StatusGet(g_motor_120_degree0.p_ctrl, &smpl_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully got current control status. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | Data received pointer is invalid. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_SpeedGet()

fsp_err_t RM_MOTOR_120_DEGREE_SpeedGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_120_DEGREE_SpeedGet(g_motor_120_degree0.p_ctrl, &smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully got rotational speed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | Data received pointer is invalid. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_DEGREE_WaitStopFlagGet ( motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag_wait_stop* )

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

```
/* Get wait stop flag */
   (void) RM_MOTOR_120_DEGREE_WaitStopFlagGet(g_motor_120_degree0.p_ctrl,
&smpl_wait_stop_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully got wait stop flag. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | Data received pointer is invalid. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_ErrorCheck()

fsp_err_t RM_MOTOR_120_DEGREE_ErrorCheck ( motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error* )

Check the occurunce of error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_120_DEGREE_ErrorCheck(g_motor_120_degree0.p_ctrl, &smpl_error);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully error checke process. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | Data received pointer is invalid. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_PositionSet()

fsp_err_t RM_MOTOR_120_DEGREE_PositionSet ( motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position* )

Set position reference. Implements motor_api_t::positionSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_AngleGet()

fsp_err_t RM_MOTOR_120_DEGREE_AngleGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad* )

Set position reference. Implements motor_api_t::angleGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_120_DEGREE_FunctionSelect()

fsp_err_t RM_MOTOR_120_DEGREE_FunctionSelect ( motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function* )

Select using function. Implements motor_api_t::functionSelect.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

## 5.2.11.6 Motor Angle (rm_motor_estimate)

Modules » Motor

**Functions**

| | |
|---|---|
| fsp_err_t | RM_MOTOR_ESTIMATE_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) |

| | |
|---|---|
| | Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_Close (motor_angle_ctrl_t *const p_ctrl)<br>Disables specified Angle Estimation module. Implements motor_angle_api_t::close. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_Reset (motor_angle_ctrl_t *const p_ctrl)<br>Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)<br>Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)<br>Set Speed Information. Implements motor_angle_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)<br>Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)<br>Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)<br>Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) |

| | |
|---|---|
| | Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_InternalCalculate (motor_angle_ctrl_t *const p_ctrl)<br>Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_AngleAdjust (motor_angle_ctrl_t *const p_ctrl)<br>Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl)<br>Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)<br>Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_CyclicProcess (motor_angle_ctrl_t *const p_ctrl)<br>Perform induction cyclic process. Implements motor_angle_api_t::cyclicProcess. More... |
| fsp_err_t | RM_MOTOR_ESTIMATE_SensorDataSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)<br>Set sensor data. Implements motor_angle_api_t::sensorDataSet. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface.

# Overview

The motor angle and speed estimation module is used to calculate rotor angle and rotational speed

in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application.

**Features**

The Motor Angle and Speed Estimation Module has below features.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

# Configuration

**Build Time Configurations for rm_motor_estimate**

The following build time configurations are defined in fsp_cfg/rm_motor_estimate_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Motor > Motor Angle and Speed Estimation (rm_motor_estimate)**

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed Estimation (rm_motor_estimate).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Motor Parameter | | | |
| Pole pairs | Must be a valid positive integer. | 2 | Pole pairs |
| Motor Parameter > Resistance[ohm] | Must be a valid value | 8.5F | Resistance |
| Motor Parameter > Inductance of d-axis[H] | Must be a valid value | 0.0045F | Inductance of d-axis |
| Motor Parameter > Inductance of q-axis[H] | Must be a valid value | 0.0045F | Inductance of q-axis |
| Motor Parameter > Permanent magnetic flux[Wb] | Must be a valid value | 0.02159F | Permanent magnetic flux |
| Motor Parameter > Rotor inertia[kgm^2] | Must be a valid value | 0.0000028F | Rotor inertia |
| Motor Parameter > Nominal current[Arms] | Must be a valid value | 1.67 | Nominal current |
| Name | Name must be a valid C symbol | g_motor_angle0 | Module name. |
| Openloop damping | • Disable<br>• Enable | Enable | Openloop damping functionally enable or |

| | | | |
|---|---|---|---|
| | | | disable |
| Natural frequency of BEMF observer | Must be a valid value | 1000.0F | Natural frequency of BEMF observer |
| Damping ratio of BEMF observer | Must be a valid value | 1.0F | Damping ratio of BEMF observer |
| Natural frequency of PLL Speed estimate loop | Must be a valid value | 20.0F | Natural frequency of PLL Speed estimate loop |
| Damping ratio of PLL Speed estimate loop | Must be a valid value | 1.0F | Damping ratio of PLL Speed estimate loop |
| Control period | Must be a valid value | 0.00005F | Control period |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Limitations

Developers should be aware of the following limitations when using the Motor Angle and Speed Estimation:

## Examples

### Basic Example

This is a basic example of minimal use of the Motor Angle and Speed Estimation in an application.

```
void motor_estimate_basic_example (void)
{
 fsp_err_t             err = FSP_SUCCESS;
 motor_angle_current_t smpl_current:
 motor_angle_voltage_reference_t smpl_voltage;
 /* Initializes the module. */
    err = RM_MOTOR_ESTIMATE_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 while (true)
```

```
    {
 /* Application work here. */
 /* Set PI Control Flag before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_FlagPiCtrlSet(&g_mtr_angle0_ctrl, 1U);
       smpl_current.id = 1.0F;
       smpl_current.iq = 1.0F;
       smpl_voltage.vd = 10.0F;
       smpl_voltage.vq = 10.0F;
 /* Set Current and Speed data before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_CurrentSet(&g_mtr_angle0_ctrl, smpl_current,
smpl_voltage);
 /* Set Internal Speed Reference & damping speed data before get Angle/Speed and
Estimated Component */
       (void) RM_MOTOR_ESTIMATE_SpeedSet(&g_mtr_angle0_ctrl, 104.27F, 10.0F);
 /* Get Angle/Speed data */
       (void) RM_MOTOR_ESTIMATE_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
 /* Get Estimated Component */
       (void) RM_MOTOR_ESTIMATE_EstimatedComponentGet(&g_mtr_angle0_ctrl, &f_get_ed,
&f_get_eq);
    }
}
```

### Function Documentation

### ◆ RM_MOTOR_ESTIMATE_Open()

fsp_err_t RM_MOTOR_ESTIMATE_Open ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MTR_ANGL_EST successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_MOTOR_ESTIMATE_Close()

fsp_err_t RM_MOTOR_ESTIMATE_Close ( motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Estimation module. Implements motor_angle_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_Reset()

fsp_err_t RM_MOTOR_ESTIMATE_Reset ( motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_CurrentSet()

fsp_err_t RM_MOTOR_ESTIMATE_CurrentSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage* )

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_SpeedSet()

fsp_err_t RM_MOTOR_ESTIMATE_SpeedSet ( motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed* )

Set Speed Information. Implements motor_angle_api_t::speedSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_ESTIMATE_FlagPiCtrlSet ( motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi* )

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_AngleSpeedGet()

fsp_err_t RM_MOTOR_ESTIMATE_AngleSpeedGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err* )

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_EstimatedComponentGet()

fsp_err_t RM_MOTOR_ESTIMATE_EstimatedComponentGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq* )

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data gotten. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_ParameterUpdate()

fsp_err_t RM_MOTOR_ESTIMATE_ParameterUpdate ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is update. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_InternalCalculate()

fsp_err_t RM_MOTOR_ESTIMATE_InternalCalculate ( motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_AngleAdjust()

fsp_err_t RM_MOTOR_ESTIMATE_AngleAdjust ( motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_EncoderCyclic()

fsp_err_t RM_MOTOR_ESTIMATE_EncoderCyclic ( motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_InfoGet()

fsp_err_t RM_MOTOR_ESTIMATE_InfoGet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info* )

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_ESTIMATE_CyclicProcess()

fsp_err_t RM_MOTOR_ESTIMATE_CyclicProcess ( motor_angle_ctrl_t *const *p_ctrl*)

Perform induction cyclic process. Implements motor_angle_api_t::cyclicProcess.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_ESTIMATE_SensorDataSet()

fsp_err_t RM_MOTOR_ESTIMATE_SensorDataSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_ad_data_t *const *p_ad_data* )

Set sensor data. Implements motor_angle_api_t::sensorDataSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

## 5.2.11.7 Motor Angle (rm_motor_sense_encoder)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_Close (motor_angle_ctrl_t *const p_ctrl)<br>Disables specified Angle Encoder module. Implements motor_angle_api_t::close. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_Reset (motor_angle_ctrl_t *const p_ctrl)<br>Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_InternalCalculate (motor_angle_ctrl_t *const p_ctrl)<br>Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)<br>Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_AngleAdjust (motor_angle_ctrl_t *const p_ctrl)<br>Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl)<br>Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_CyclicProcess (motor_angle_ctrl_t *const p_ctrl)<br>Perform cyclic process. Implements motor_angle_api_t::cyclicProcess. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)<br>Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)<br>Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)<br>Set Speed Information. Implements motor_angle_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)<br>Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)<br>Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More... |
| fsp_err_t | RM_MOTOR_SENSE_ENCODER_SensorDataSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)<br>Set sensor data. Implements motor_angle_api_t::sensorDataSet. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface.

# Overview

The motor angle and speed calculation with an encoder module is used to calculate rotor angle and rotational speed in an application. This module is designed to be used with the motor current module (rm_motor_current).

### Features

The motor angle and speed calculation with an encoder module has the features listed below.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

# Configuration

### Build Time Configurations for rm_motor_sense_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_sense_encoder_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support HALL sensor | • Support<br>• Unsupport | Support | |

### Configurations for Motor > Motor Angle and Speed Calculation with encoder (rm_motor_sense_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed Calculation with encoder (rm_motor_sense_encoder).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_sense_encoder0 | Module name. |
| Hall sensor | | | |
| sensor input port U | Manual Entry | BSP_IO_PORT_09_PIN_07 | U-phase Hall sensor input port |
| sensor input port V | Manual Entry | BSP_IO_PORT_09_PIN_05 | V-phase Hall sensor input port |
| sensor input port W | Manual Entry | BSP_IO_PORT_09_PIN_06 | W-phase Hall sensor input port |
| sensor pattern #1 | Must be a valid integer | 1 | Hall sensor pattern #1 |
| sensor pattern #2 | Must be a valid integer | 5 | Hall sensor pattern #2 |
| sensor pattern #3 | Must be a valid integer | 4 | Hall sensor pattern #3 |
| sensor pattern #4 | Must be a valid integer | 6 | Hall sensor pattern #4 |
| sensor pattern #5 | Must be a valid integer | 2 | Hall sensor pattern #5 |
| sensor pattern #6 | Must be a valid integer | 3 | Hall sensor pattern #6 |
| Motor Parameter | | | |

| | | | |
|---|---|---|---|
| Pole pairs | Must be a valid positive integer. | 7 | Pole pairs |
| Resistance (ohm) | Must be a valid non-negative value | 0.453F | Resistance |
| Inductance of d-axis (H) | Must be a valid non-negative value | 0.0009447F | Inductance of d-axis |
| Inductance of q-axis (H) | Must be a valid non-negative value | 0.0009447F | Inductance of q-axis |
| Permanent magnetic flux (Wb) | Must be a valid non-negative value | 0.006198F | Permanent magnetic flux |
| Motor Parameter > Rotor inertia (kgm^2) | Must be a valid non-negative value | 0.00000962F | Rotor inertia |
| Control Type | • Speed<br>• Position | Position | Select control Type |
| Period of Current control (kHz) | Must be a valid non-negative value | 20.0F | Period of Current control |
| Period of Speed control (sec) | Must be a valid non-negative value | 0.0005F | Period of Speed control |
| PWM Carrier Frequency (kHz) | Must be a valid non-negative value | 20.0F | PWM Carrier Frequency |
| Decimation of Interrupt | Must be a valid positive integer. | 0U | Decimation of Interrupt |
| Counts per Rotation | Must be a valid positive integer. | 1200U | Encoder Counts per One Rotation |
| Counts for Angle Adjust | Must be a valid positive integer. | 512U | Counts for Angle Adjust (as working time) |
| Zero speed counts | Must be a valid positive integer. | 20000000U | Threshold counts to judge zero speed |
| Occupancy Time | Must be a valid non-negative value | 0.30F | Occupancy time of carrier interrup |
| Carrier Time | Must be a valid non-negative value | 0.000013F | Processing time of carrier interrupt |
| Process Time | Must be a valid non-negative value | 0.000001F | Processing time of encoder interrupt |
| Highspeed Change Margin (rpm) | Must be a valid positive integer. | 150U | Margin of toggle speed for high speed mode |
| LPF parameter for Highspeed Filter | Must be a valid non-negative value | 0.1F | Highspeed mode speed LPF parameter |
| Counts to change speed | Must be a valid positive integer. | 8U | Counts for mode change of position speed calculation |
| Speed Detection Type | • Use capture | Use capture width | Select Speed detection |

| | | |
|---|---|---|
| | width<br>• Use carrier interrupt | Type |

### Clock Configuration

### Pin Configuration

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the motor angle and speed calculation with an encoder: all configurations should be set as positive values.

# Examples

### Basic Example

This is a basic example of minimal use of the motor angle and speed calculation with an encoder in an application.

```
void motor_sense_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSE_ENCODER_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Initialize motor with the encoder */
       (void) RM_MOTOR_SENSE_ENCODER_AngleAdjust(&g_mtr_angle0_ctrl);
 /* Perform cyclic encoder process*/
       (void) RM_MOTOR_SENSE_ENCODER_CyclicProcess(&g_mtr_angle0_ctrl);
 /* Calculate information with encoder signal input */
       (void) RM_MOTOR_SENSE_ENCODER_InternalCalculate(&g_mtr_angle0_ctrl);
 /* Get angle/speed data */
       (void) RM_MOTOR_SENSE_ENCODER_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
```

```
/* Get calculated component */
      (void) RM_MOTOR_SENSE_ENCODER_InfoGet(&g_mtr_angle0_ctrl, &temp_info);
   }
/* Reset the module */
   (void) RM_MOTOR_SENSE_ENCODER_Reset(&g_mtr_angle0_ctrl);
/* Close the module */
   (void) RM_MOTOR_SENSE_ENCODER_Close(&g_mtr_angle0_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_sense_encoder_loop_t |
| enum | motor_sense_encoder_mode_t |
| enum | motor_sense_encoder_speed_detect_type_t |

## Enumeration Type Documentation

### ◆ motor_sense_encoder_loop_t

| enum motor_sense_encoder_loop_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_ENCODER_LOOP_SPEED | Speed control mode. |
| MOTOR_SENSE_ENCODER_LOOP_POSITION | Position control mode. |

### ◆ motor_sense_encoder_mode_t

| enum motor_sense_encoder_mode_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_ENCODER_MODE_INIT | Initialize mode (Start status) |
| MOTOR_SENSE_ENCODER_MODE_BOOT | Boot mode (Angle adjustment status) |
| MOTOR_SENSE_ENCODER_MODE_DRIVE | Drive mode (Normal work status) |

### ◆ motor_sense_encoder_speed_detect_type_t

| enum motor_sense_encoder_speed_detect_type_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_ENCODER_SPEED_DETECT_WIDTH | Use captured width. |
| MOTOR_SENSE_ENCODER_SPEED_DETECT_CARRIER | Use carrier interrupt. |

## Function Documentation

### ◆ RM_MOTOR_SENSE_ENCODER_Open()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Open ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Angle Encoder module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_MOTOR_SENSE_ENCODER_Close()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Close ( motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Encoder module. Implements motor_angle_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_Reset()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Reset ( motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_InternalCalculate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InternalCalculate ( motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_AngleSpeedGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleSpeedGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err* )

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_AngleAdjust()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleAdjust ( motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_EncoderCyclic()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EncoderCyclic ( motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_CyclicProcess()

fsp_err_t RM_MOTOR_SENSE_ENCODER_CyclicProcess ( motor_angle_ctrl_t *const *p_ctrl*)

Perform cyclic process. Implements motor_angle_api_t::cyclicProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_InfoGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InfoGet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info* )

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_ParameterUpdate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_ParameterUpdate ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is update. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_ENCODER_CurrentSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_CurrentSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage* )

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense encoder software currentSet is not supported. |

### ◆ RM_MOTOR_SENSE_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_SpeedSet ( motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed* )

Set Speed Information. Implements motor_angle_api_t::speedSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense encoder software speedSet is not supported. |

### ◆ RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet ( motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi* )

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense encoder software flagPiCtrlSet is not supported. |

### ◆ RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq* )

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense encoder software estimatedComponentGet is not supported. |

### ◆ RM_MOTOR_SENSE_ENCODER_SensorDataSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_SensorDataSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_ad_data_t *const *p_ad_data* )

Set sensor data. Implements motor_angle_api_t::sensorDataSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### 5.2.11.8 Motor Angle and Speed Calculation with Hall sensors (rm_motor_sense_hall)

Modules » Motor

#### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_HALL_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Opens and configures the angle hall sensor module. Implements motor_angle_api_t::open. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_Close (motor_angle_ctrl_t *const p_ctrl)<br>Disables specified Angle Estimation module. Implements motor_angle_api_t::close. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_Reset (motor_angle_ctrl_t *const p_ctrl)<br>Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)<br>Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)<br>Set Speed Information. Implements motor_angle_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)<br>Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)<br>Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_HALL_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)<br>Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_InternalCalculate (motor_angle_ctrl_t *const p_ctrl)<br>Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_AngleAdjust (motor_angle_ctrl_t *const p_ctrl)<br>Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl)<br>Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More... |
| fsp_err_t | RM_MOTOR_SENSE_HALL_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)<br>Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface.

# Overview

The motor angle and speed calculation with hall sensors module is used to calculate rotor angle and rotational speed in an application. This module is designed to be used with the motor current module (rm_motor_current).

### Features

The motor angle and speed calculation with hall sensors module has the features listed below.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

# Configuration

### Build Time Configurations for rm_motor_sense_hall

The following build time configurations are defined in fsp_cfg/rm_motor_sense_hall_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Angle and Speed Calculation with Hall sensors (rm_motor_sense_hall)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed Calculation with Hall sensors (rm_motor_sense_hall).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_angle0 | Module name. |
| Hall sensor | | | |
| U phase input port | Manual Entry | BSP_IO_PORT_12_PIN_04 | Hall sensor port U |
| V phase input port | Manual Entry | BSP_IO_PORT_12_PIN_05 | Hall sensor port V |
| W phase input port | Manual Entry | BSP_IO_PORT_11_PIN_01 | Hall sensor port W |
| sensor pattern #1 | Must be a valid integer | 1 | Hall sensor pattern #1 |
| sensor pattern #2 | Must be a valid integer | 5 | Hall sensor pattern #2 |
| sensor pattern #3 | Must be a valid integer | 4 | Hall sensor pattern #3 |
| sensor pattern #4 | Must be a valid integer | 6 | Hall sensor pattern #4 |
| sensor pattern #5 | Must be a valid integer | 2 | Hall sensor pattern #5 |
| sensor pattern #6 | Must be a valid integer | 3 | Hall sensor pattern #6 |
| Process Frequency (kHz) | Must be a valid value | 20.0F | Frequency to perform the process. |
| Correction parameter | Must be a valid value | 0.0F | Correction parameter |

| | | | |
|---|---|---|---|
| of rotor angle | | | of rotor angle |
| Default counts of carrier interrupt | Must be a valid non-negative value. | 300U | Default counts of carrier interrupt during a period of Hall signal change |
| Maximum counts of one rotation | Must be a valid non-negative value. | 500U | Maximum counts of carrier interrupt during one rotor rotation |
| Target value for pseudo speed (rad/s) | Must be a valid non-negative value. | 100.0 | Target value for pseudo speed (rad/s). |
| Target time until the pseudo speed update reaches (msec) | Must be a valid non-negative value. | 300.0 | Target time until the pseudo speed update reaches (msec). |
| Rotation counts to start speed estimation | Must be a valid integer | 2 | Rotation counts to start of speed estimation. |
| Carrier counts at startup | Must be a valid integer | 400 | Carrier counts to wait the start timing of pseudo speed update |
| Speed to judge start | Must be a valid non-negative value. | 250.0 | Speed to judge start PI calculation. |

**Clock Configuration**

**Pin Configuration**

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the motor angle and speed calculation with hall sensors: all configurations should be set as positive values.

# Examples

### Basic Example

This is a basic example of minimal use of the motor angle and speed calculation with hall sensors in an application.

```
void motor_sense_hall_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSE_HALL_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
```

```
 * This implementation is an example. */
// while (true)
   {
/* Application work here. */
/* Get angle/speed data */
      (void) RM_MOTOR_SENSE_HALL_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
   }
/* Reset the module */
   (void) RM_MOTOR_SENSE_HALL_Reset(&g_mtr_angle0_ctrl);
/* Close the module */
   (void) RM_MOTOR_SENSE_HALL_Close(&g_mtr_angle0_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | motor_sense_hall_input_t |
| struct | motor_sense_hall_extended_cfg_t |
| struct | motor_sense_hall_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | motor_sense_hall_direction_t |
| enum | motor_sense_hall_signal_status_t |

## Data Structure Documentation

### ◆ motor_sense_hall_input_t

| struct motor_sense_hall_input_t | | |
|---|---|---|
| This structure is provided to receive speed information. | | |
| Data Fields | | |
| float | f4_ref_speed_rad_ctrl | Speed Reference [rad/sec]. |

### ◆ motor_sense_hall_extended_cfg_t

| struct motor_sense_hall_extended_cfg_t | | |
|---|---|---|
| Optional Motor sense hall extension data structure. | | |
| Data Fields | | |
| bsp_io_port_pin_t | port_hall_sensor_u | Hall U-signal input port. |

| | | |
|---|---|---|
| bsp_io_port_pin_t | port_hall_sensor_v | Hall V-signal input port. |
| bsp_io_port_pin_t | port_hall_sensor_w | Hall W-signal input port. |
| uint8_t | u1_hall_pattern[MOTOR_SENSE_HALL_SPEED_COUNTS+1] | The order of hall signal pattern. |
| float | f_pwm_carrier_freq | PWM carrier frequency (or Decimated frequency at decimation of current process) |
| float | f_angle_correct | Coefficient to correct angle. |
| uint8_t | u1_trigger_hall_signal_count | Rotation counts to wait the stability. |
| float | f4_target_pseudo_speed_rad | Target value for pseudo speed estimates [radian/second]. |
| float | f4_reach_time_msec | Time until the pseudo speed estimate reaches the target value [msec]. |
| uint16_t | u2_trigger_carrier_count | Estimated speed 0 until this trigger. |
| uint16_t | u2_default_counts | Default counts for period of hall signal to reset. |
| uint16_t | u2_maximum_period | Maximum counts of hall signal period. |
| uint8_t | u1_hall_polepairs | Hall pole pairs. |
| float | f4_start_speed_rad | Speed to judge start [radian/second]. |

### ◆ motor_sense_hall_instance_ctrl_t

| struct motor_sense_hall_instance_ctrl_t | | |
|---|---|---|
| SENSE_HALL control block. DO NOT INITIALIZE. Initialization occurs when motor_angle_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | |
| uint8_t | u1_hall_signal | Hall signal pattern. |
| uint8_t | u1_last_hall_signal | Last hall signal pattern. |
| motor_sense_hall_direction_t | direction | Rotation direction. |
| motor_sense_hall_direction_t | last_direction | Last rotation direction. |
| uint16_t | u2_carrier_count | Carrier count. |
| uint16_t | u2_hall_period[MOTOR_SENSE_HALL_SPEED_COUNTS] | Array of carrier count to calculate 2PI. |
| uint8_t | u1_period_counter | Counter for above array. |
| | | |

| | | |
|---|---|---|
| float | f_angle | Rotor angle [radian]. |
| float | f_angle_per_count | Angle per 1 count. |
| float | f_calculated_speed | Calculated speed [radian/second]. |
| uint8_t | u1_hall_signal_memory | Memorized hall signal at startup. |
| motor_sense_hall_signal_status_t | hall_signal_status | Hall signal status. |
| uint8_t | u1_hall_signal_count | Rotation counter. |
| float | f4_pseudo_speed_rad | Pseudo speed used for startup [radian/second]. |
| float | f4_add_pseudo_speed_rad | Step of pseudo speed to update [radian/second]. |
| uint16_t | u2_startup_carrier_count | Counter of carrier interrupt for startup. |
| motor_sense_hall_input_t | st_input | Input parameter structure. |
| uint8_t | u1_startup_flag | Flag for startup. |
| motor_angle_cfg_t const * | p_cfg | |

## Enumeration Type Documentation

### ◆ motor_sense_hall_direction_t

| enum motor_sense_hall_direction_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_HALL_DIRECTION_CW | Rotation direction clockwise. |
| MOTOR_SENSE_HALL_DIRECTION_CCW | Rotation direction counter clockwise. |

### ◆ motor_sense_hall_signal_status_t

| enum motor_sense_hall_signal_status_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_HALL_SIGNAL_STATUS_INITIAL | Hall signal isn't captured. (Initial) |
| MOTOR_SENSE_HALL_SIGNAL_STATUS_CAPTURED | Hall signal is captured. |

## Function Documentation

### ◆ RM_MOTOR_SENSE_HALL_Open()

fsp_err_t RM_MOTOR_SENSE_HALL_Open ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Opens and configures the angle hall sensor module. Implements motor_angle_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MTR_ANGL_EST successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ RM_MOTOR_SENSE_HALL_Close()

fsp_err_t RM_MOTOR_SENSE_HALL_Close ( motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Estimation module. Implements motor_angle_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_HALL_Reset()

fsp_err_t RM_MOTOR_SENSE_HALL_Reset ( motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_HALL_CurrentSet()

fsp_err_t RM_MOTOR_SENSE_HALL_CurrentSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage* )

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_SpeedSet()

fsp_err_t RM_MOTOR_SENSE_HALL_SpeedSet ( motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed* )

Set Speed Information. Implements motor_angle_api_t::speedSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data get successfully. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_HALL_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_SENSE_HALL_FlagPiCtrlSet ( motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi* )

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_AngleSpeedGet()

fsp_err_t RM_MOTOR_SENSE_HALL_AngleSpeedGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err* )

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is invalid. |

### ◆ RM_MOTOR_SENSE_HALL_EstimatedComponentGet()

fsp_err_t RM_MOTOR_SENSE_HALL_EstimatedComponentGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq* )

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_ParameterUpdate()

fsp_err_t RM_MOTOR_SENSE_HALL_ParameterUpdate ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is update. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_HALL_InternalCalculate()

fsp_err_t RM_MOTOR_SENSE_HALL_InternalCalculate ( motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_AngleAdjust()

fsp_err_t RM_MOTOR_SENSE_HALL_AngleAdjust ( motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_EncoderCyclic()

fsp_err_t RM_MOTOR_SENSE_HALL_EncoderCyclic ( motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_MOTOR_SENSE_HALL_InfoGet()

fsp_err_t RM_MOTOR_SENSE_HALL_InfoGet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info* )

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

## 5.2.11.9 Motor Angle and Speed Calculation with induction sensor (rm_motor_sense_induction)

Modules » Motor

## Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Opens and configures the Angle module. Implements motor_angle_api_t::open. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_Close (motor_angle_ctrl_t *const p_ctrl)<br>Disables specified Angle module. Implements motor_angle_api_t::close. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_Reset (motor_angle_ctrl_t *const p_ctrl)<br>Reset variables of Angle module. Implements motor_angle_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_InternalCalculate (motor_angle_ctrl_t *const p_ctrl)<br>Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)<br>Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_AngleAdjust (motor_angle_ctrl_t *const p_ctrl)<br>Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl)<br>Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_CyclicProcess (motor_angle_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | Induction sensor Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::cyclicProcess. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)<br>Gets information of Angle Module. Implements motor_angle_api_t::infoGet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_CorrectReset (motor_angle_ctrl_t *const p_ctrl)<br>Reset to restart calibration Angle module. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_ErrorGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_error_t *const p_error)<br>Gets the error information about induction correction. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)<br>Update the parameters of angle & speed calculation. Implements motor_angle_api_t::parameterUpdate. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_SensorDataSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)<br>Set A/D Converted Data. Implements motor_angle_api_t::sensorDataSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)<br>Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)<br>Set Speed Information. Implements motor_angle_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) |

| | |
|---|---|
| | Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More... |
| fsp_err_t | RM_MOTOR_SENSE_INDUCTION_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)<br>Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor angle Interface.

# Overview

The motor angle and speed calculation with induction sensor module is used to calculate rotor angle and rotational speed in an application. This module is designed to be used with the motor current module (rm_motor_current).

### Features

The motor angle and speed calculation with induction sensor module has the features listed below.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

# Configuration

### Build Time Configurations for rm_motor_sense_induction

The following build time configurations are defined in fsp_cfg/rm_motor_sense_induction_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Angle and Speed Calculation with induction sensor (rm_motor_sense_induction)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle and Speed Calculation with induction sensor (rm_motor_sense_induction).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_angle0 | Module name. |

| | | | |
|---|---|---|---|
| Control type | • Speed<br>• Position | Position | Select control Type |
| Frequency of current control (kHz) | Must be a valid non-negative value. | 20.0 | Frequency of current control |
| Decimation of Interrupt | Must be a valid integer. | 1 | Decimation of Interrupt |
| Counts to get signal | Must be a valid integer. | 10 | Counts to get analog input signal |
| Limit of signal error | Must be a valid integer. | 100 | Limit of signal error |
| Coefficient of speed LPF | Must be a valid value | 0.07 | Coefficient of speed LPF |
| A/D reference voltage | Must be a valid non-negative value. | 3.3 | Reference voltage of A/D converter |
| A/D conversion scale | Must be a valid non-negative value. | 4095.0 | Conversion scale of A/D converter |
| Openloop speed (rpm) | Must be a valid non-negative value. | 6.0 | Rotation speed at calibration openloop (rpm) |
| D-axis current at openloop (A) | Must be a valid non-negative value. | 1.0 | D-axis current at calibration openloop (A) |
| Angle adjustment times | Must be a valid integer. | 512 | Times to get signal at angle adjustment |
| Calibration enable | • Disable<br>• Enable | Enable | Select enable/disable of signal calibration |
| Induction sensor pole pairs | Must be a valid integer. | 4 | Induction sensor pole pairs |
| Motor pole pairs | Must be a valid integer. | 4 | Motor pole pairs |

**Clock Configuration**

**Pin Configuration**

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the motor angle and speed calculation with induction sensor: all configurations should be set as positive values.

# Examples

### Basic Example

This is a basic example of minimal use of the motor angle and speed calculation with induction sensor in an application.

```
void motor_sense_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSE_INDUCTION_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Initialize motor */
       (void) RM_MOTOR_SENSE_INDUCTION_AngleAdjust(&g_mtr_angle0_ctrl);
 /* Perform cyclic process*/
       (void) RM_MOTOR_SENSE_INDUCTION_CyclicProcess(&g_mtr_angle0_ctrl);
 /* Calculate information with induction sensor signal input */
       (void) RM_MOTOR_SENSE_INDUCTION_InternalCalculate(&g_mtr_angle0_ctrl);
 /* Get angle/speed data */
       (void) RM_MOTOR_SENSE_INDUCTION_AngleSpeedGet(&g_mtr_angle0_ctrl,
&f_get_angle, &f_get_speed, &f_get_phase_err);
    }
 /* Reset the module */
    (void) RM_MOTOR_SENSE_INDUCTION_Reset(&g_mtr_angle0_ctrl);
 /* Close the module */
    (void) RM_MOTOR_SENSE_INDUCTION_Close(&g_mtr_angle0_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_sense_induction_loop_t |
| enum | motor_sense_induction_calibration_t |
| enum | motor_sense_induction_mode_t |

## Enumeration Type Documentation

### ◆ motor_sense_induction_loop_t

| enum motor_sense_induction_loop_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_INDUCTION_LOOP_SPEED | Speed control mode. |
| MOTOR_SENSE_INDUCTION_LOOP_POSITION | Position control mode. |

### ◆ motor_sense_induction_calibration_t

| enum motor_sense_induction_calibration_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_INDUCTION_CALIBRATION_DISABLE | Disable calibration. |
| MOTOR_SENSE_INDUCTION_CALIBRATION_ENABLE | Enable calibration. |

### ◆ motor_sense_induction_mode_t

| enum motor_sense_induction_mode_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_INDUCTION_MODE_INIT | Initialize mode (Start status) |
| MOTOR_SENSE_INDUCTION_MODE_BOOT | Boot mode (Angle adjustment status) |
| MOTOR_SENSE_INDUCTION_MODE_DRIVE | Drive mode (Normal work status) |

## Function Documentation

### ◆ RM_MOTOR_SENSE_INDUCTION_Open()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_Open ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Opens and configures the Angle module. Implements motor_angle_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Angle Induction module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_MOTOR_SENSE_INDUCTION_Close()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_Close ( motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle module. Implements motor_angle_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_Reset()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_Reset ( motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle module. Implements motor_angle_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_InternalCalculate()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_InternalCalculate ( motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_AngleSpeedGet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_AngleSpeedGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err* )

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_AngleAdjust()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_AngleAdjust ( motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_EncoderCyclic()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_EncoderCyclic ( motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense induction software encoderCyclic is not supported. |

### ◆ RM_MOTOR_SENSE_INDUCTION_CyclicProcess()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_CyclicProcess ( motor_angle_ctrl_t *const *p_ctrl*)

Induction sensor Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::cyclicProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_InfoGet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_InfoGet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info* )

Gets information of Angle Module. Implements motor_angle_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_CorrectReset()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_CorrectReset ( motor_angle_ctrl_t *const *p_ctrl*)

Reset to restart calibration Angle module.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_ErrorGet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_ErrorGet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_error_t *const *p_error* )

Gets the error information about induction correction.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data calculated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_ParameterUpdate()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_ParameterUpdate ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg* )

Update the parameters of angle & speed calculation. Implements motor_angle_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is update. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_SensorDataSet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_SensorDataSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_ad_data_t *const *p_ad_data* )

Set A/D Converted Data. Implements motor_angle_api_t::sensorDataSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is update. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SENSE_INDUCTION_CurrentSet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_CurrentSet ( motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage* )

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense induction software currentSet is not supported. |

### ◆ RM_MOTOR_SENSE_INDUCTION_SpeedSet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_SpeedSet ( motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed* )

Set Speed Information. Implements motor_angle_api_t::speedSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense induction software speedSet is not supported. |

### ◆ RM_MOTOR_SENSE_INDUCTION_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_FlagPiCtrlSet ( motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi* )

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense induction software flagPiCtrlSet is not supported. |

### ◆ RM_MOTOR_SENSE_INDUCTION_EstimatedComponentGet()

fsp_err_t RM_MOTOR_SENSE_INDUCTION_EstimatedComponentGet ( motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq* )

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Motor sense induction software estimatedComponentGet is not supported. |

## 5.2.11.10 Motor Current Controller (rm_motor_current)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_CURRENT_Open (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)<br>Opens and configures the Motor Current Module. Implements motor_current_api_t::open. More... |
| fsp_err_t | RM_MOTOR_CURRENT_Close (motor_current_ctrl_t *const p_ctrl)<br>Disables specified Motor Current Module. Implements motor_current_api_t::close. More... |
| fsp_err_t | RM_MOTOR_CURRENT_Reset (motor_current_ctrl_t *const p_ctrl)<br>Reset variables of Motor Current Module. Implements motor_current_api_t::reset. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_CURRENT_Run (motor_current_ctrl_t *const p_ctrl)<br>Run(Start) the Current Control. Implements motor_current_api_t::run. More... |
| fsp_err_t | RM_MOTOR_CURRENT_ParameterSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)<br>Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_CurrentReferenceSet (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)<br>Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_SpeedPhaseSet (motor_current_ctrl_t *const p_ctrl, float const speed, float const phase)<br>Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_CurrentSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)<br>Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_ParameterGet (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)<br>Get Output Parameters. Implements motor_current_api_t::parameterGet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_CurrentGet (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)<br>Get d/q-axis Current. Implements motor_current_api_t::currentGet. More... |
| fsp_err_t | RM_MOTOR_CURRENT_PhaseVoltageGet (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage)<br>Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_CURRENT_ParameterUpdate (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)<br>Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate. More... |
| void | rm_motor_current_encoder_cyclic (motor_current_instance_t const *p_ctrl) |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor current Interface.

# Overview

The motor current is used to control the electric current of motor rotation in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

### Features

The Motor Current Module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling Control.
- Voltage Error Compensation.

# Configuration

### Build Time Configurations for rm_motor_current

The following build time configurations are defined in fsp_cfg/rm_motor_current_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Current Controller (rm_motor_current)

This module can be added to the Stacks tab via New Stack > Motor > Motor Current Controller (rm_motor_current).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_current0 | Module name. |

| | | | |
|---|---|---|---|
| Sensor type | • Sensorless<br>• Encoder<br>• Induction<br>• Hall | Sensorless | Select sensor type |
| Shunt type | • 1 shunt<br>• 2 shunt<br>• 3 shunt | 2 shunt | Select shunt type |
| Current control decimation | Must be a valid non-negative value. | 0 | Decimation of current control. |
| PWM carrier frequency (kHz) | Must be a valid value | 20.0F | PWM carrier frequency. |
| Input voltage (V) | Must be a valid value | 24.0F | Input voltage for limitation of current PI control. |
| Sample delay compensation | • Disable<br>• Enable | Enable | Select enable/disable sample delay compensation. |
| Period magnification value | Must be a valid value | 1.5 | Period magnification value for sampling delay compensation. |
| Voltage error compensation | • Disable<br>• Enable | Enable | Select enable/disable voltage error compensation. |
| Voltage error compensation table of voltage 1 | Must be a valid value | 0.672F | Voltage error compensation table of voltage. |
| Voltage error compensation table of voltage 2 | Must be a valid value | 0.945F | Voltage error compensation table of voltage. |
| Voltage error compensation table of voltage 3 | Must be a valid value | 1.054F | Voltage error compensation table of voltage. |
| Voltage error compensation table of voltage 4 | Must be a valid value | 1.109F | Voltage error compensation table of voltage. |
| Voltage error compensation table of voltage 5 | Must be a valid value | 1.192F | Voltage error compensation table of voltage. |
| Voltage error compensation table of current 1 | Must be a valid value | 0.013F | Voltage error compensation table of current. |
| Voltage error compensation table of current 2 | Must be a valid value | 0.049F | Voltage error compensation table of current. |
| Voltage error | Must be a valid value | 0.080F | Voltage error |

| | | | |
|---|---|---|---|
| compensation table of current 3 | | | compensation table of current. |
| Voltage error compensation table of current 4 | Must be a valid value | 0.184F | Voltage error compensation table of current. |
| Voltage error compensation table of current 5 | Must be a valid value | 0.751F | Voltage error compensation table of current. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt. |
| Design Parameter | | | |
| Current PI loop omega (Hz) | Must be a valid value | 300.0F | Current PI loop omega |
| Current PI loop zeta | Must be a valid value | 1.0F | Current PI loop zeta |
| Motor Parameter | | | |
| Pole pairs | Must be a valid non-negative value. | 2 | Pole pairs |
| Resistance (ohm) | Must be a valid value | 8.5F | Resistance |
| Inductance of d-axis (H) | Must be a valid value | 0.0045F | Inductance of d-axis |
| Inductance of q-axis (H) | Must be a valid value | 0.0045F | Inductance of q-axis |
| Permanent magnetic flux (Wb) | Must be a valid value | 0.02159F | Permanent magnetic flux |
| Motor Parameter > Rotor inertia (kgm^2) | Must be a valid value | 0.0000028F | Rotor inertia |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Limitations

- Set the Period of Current Control with none-negative value.
- Set the Reference Voltage with none-negative value.

# Examples

### Basic Example

This is a basic example of minimal use of the Motor Current in an application.

```
void motor_current_basic_example (void)
{
 motor_current_input_current_t temp_input_current;
 motor_current_input_voltage_t temp_input_voltage;
 motor_current_get_voltage_t   temp_get_voltage;
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_CURRENT_Open(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set current reference before get phase voltage */
       (void) RM_MOTOR_CURRENT_CurrentReferenceSet(g_test_motor_current.p_ctrl, 1.0F,
1.0F);
 /* Set speed and phase data before get phase voltage */
       (void) RM_MOTOR_CURRENT_SpeedPhaseSet(g_test_motor_current.p_ctrl, 104.72F,
1.0F);
       temp_input_current.iu    = 1.0F;
       temp_input_current.iv    = 1.0F;
       temp_input_current.iw    = 1.0F;
       temp_input_voltage.vdc    = 24.0F;
       temp_input_voltage.va_max = 24.0F;
 /* Set electric current and voltage before get phase voltage */
       (void) RM_MOTOR_CURRENT_CurrentSet(g_test_motor_current.p_ctrl,
temp_input_current, temp_input_voltage);
 /* Activate the process. */
```

```
      (void) RM_MOTOR_CURRENT_Run(g_test_motor_current.p_ctrl);
 /* Get d/q-axis current*/
      (void) RM_MOTOR_CURRENT_CurrentGet(g_test_motor_current.p_ctrl, &f_get_id,
&f_get_iq);
 /* Get the flag of PI control */
      (void) RM_MOTOR_CURRENT_PhaseVolageGet(g_test_motor_current.p_ctrl,
&temp_get_voltage);
 /* Get Output Parameter */
      (void) RM_MOTOR_CURRENT_ParameterGet(g_test_motor_current.p_ctrl,
&test_output);
      (void) RM_MOTOR_CURRENT_ParameterUpdate(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    }
 /* Reset the process. */
    (void) RM_MOTOR_CURRENT_Reset(g_test_motor_current.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_CURRENT_Close(g_test_motor_current.p_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_current_shunt_type_t |

## Enumeration Type Documentation

### ◆ motor_current_shunt_type_t

| enum motor_current_shunt_type_t | |
|---|---|
| Selection of shunt type | |
| Enumerator | |
| MOTOR_CURRENT_SHUNT_TYPE_1_SHUNT | Only use U phase current. |
| MOTOR_CURRENT_SHUNT_TYPE_2_SHUNT | Use U and W phase current. |
| MOTOR_CURRENT_SHUNT_TYPE_3_SHUNT | Use all phase current. |

## Function Documentation

### ◆ RM_MOTOR_CURRENT_Open()

fsp_err_t RM_MOTOR_CURRENT_Open ( motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg* )

Opens and configures the Motor Current Module. Implements motor_current_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor Current successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_MOTOR_CURRENT_Close()

fsp_err_t RM_MOTOR_CURRENT_Close ( motor_current_ctrl_t *const *p_ctrl*)

Disables specified Motor Current Module. Implements motor_current_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_CURRENT_Reset()

fsp_err_t RM_MOTOR_CURRENT_Reset ( motor_current_ctrl_t *const *p_ctrl*)

Reset variables of Motor Current Module. Implements motor_current_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_CURRENT_Run()

fsp_err_t RM_MOTOR_CURRENT_Run ( motor_current_ctrl_t *const *p_ctrl*)

Run(Start) the Current Control. Implements motor_current_api_t::run.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully run. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_CURRENT_ParameterSet()

fsp_err_t RM_MOTOR_CURRENT_ParameterSet ( motor_current_ctrl_t *const *p_ctrl*, motor_current_input_t const *const *p_st_input* )

Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input argument error. |

### ◆ RM_MOTOR_CURRENT_CurrentReferenceSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentReferenceSet ( motor_current_ctrl_t *const *p_ctrl*, float const *id_reference*, float const *iq_reference* )

Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_CURRENT_SpeedPhaseSet()

fsp_err_t RM_MOTOR_CURRENT_SpeedPhaseSet ( motor_current_ctrl_t *const *p_ctrl*, float const *speed*, float const *phase* )

Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_CURRENT_CurrentSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentSet ( motor_current_ctrl_t *const *p_ctrl*, motor_current_input_current_t const *const *p_st_current*, motor_current_input_voltage_t const *const *p_st_voltage* )

Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_CURRENT_ParameterGet()

fsp_err_t RM_MOTOR_CURRENT_ParameterGet ( motor_current_ctrl_t *const *p_ctrl*, motor_current_output_t *const *p_st_output* )

Get Output Parameters. Implements motor_current_api_t::parameterGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_CURRENT_CurrentGet()

fsp_err_t RM_MOTOR_CURRENT_CurrentGet ( motor_current_ctrl_t *const *p_ctrl*, float *const *p_id*, float *const *p_iq* )

Get d/q-axis Current. Implements motor_current_api_t::currentGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data get. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_CURRENT_PhaseVoltageGet()

fsp_err_t RM_MOTOR_CURRENT_PhaseVoltageGet ( motor_current_ctrl_t *const *p_ctrl*, motor_current_get_voltage_t *const *p_voltage* )

Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data calculation. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_CURRENT_ParameterUpdate()

fsp_err_t RM_MOTOR_CURRENT_ParameterUpdate ( motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg* )

Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ rm_motor_current_encoder_cyclic()

| void rm_motor_current_encoder_cyclic ( motor_current_instance_t const * *p_ctrl*) |
|---|
| (end addtogroup MOTOR_CURRENT) |

## 5.2.11.11 Motor Encoder Vector Control (rm_motor_encoder)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_ENCODER_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MOTOR_ENCODER_Close (motor_ctrl_t *const p_ctrl)<br>Disables specified Motor Encoder Control block. Implements motor_api_t::close. More... |
| fsp_err_t | RM_MOTOR_ENCODER_Reset (motor_ctrl_t *const p_ctrl)<br>Reset Motor Encoder Control block. Implements motor_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_ENCODER_Run (motor_ctrl_t *const p_ctrl)<br>Run Motor (Start motor rotation). Implements motor_api_t::run. More... |
| fsp_err_t | RM_MOTOR_ENCODER_Stop (motor_ctrl_t *const p_ctrl)<br>Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_ENCODER_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error)<br>Set error information. Implements motor_api_t::errorSet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed reference[rpm]. Implements motor_api_t::speedSet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_ENCODER_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)<br>Set position reference[degree]. Implements motor_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)<br>Get current control status. Implements motor_api_t::statusGet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)<br>Get current rotor angle. Implements motor_api_t::angleGet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get rotational speed. Implements motor_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)<br>Check the occurunce of Error. Implements motor_api_t::errorCheck. More... |
| fsp_err_t | RM_MOTOR_ENCODER_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)<br>Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_ENCODER_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)<br>Select using function. Implements motor_api_t::functionSelect. More... |
| fsp_err_t | RM_MOTOR_ENCODER_InertiaEstimateStart (motor_ctrl_t *const p_ctrl)<br>Start inertia estimation function. More... |
| fsp_err_t | RM_MOTOR_ENCODER_InertiaEstimateStop (motor_ctrl_t *const p_ctrl)<br>Stop(Cancel) inertia estimation function. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_ENCODER_ReturnOriginStart (motor_ctrl_t *const p_ctrl)<br>Start return origin function. More... |
| fsp_err_t | RM_MOTOR_ENCODER_ReturnOriginStop (motor_ctrl_t *const p_ctrl)<br>Stop(Cancel) return origin function. More... |

### Detailed Description

Control a SPM motor on RA MCUs. This module implements the Motor Encoder Vector Control (rm_motor_encoder).

# Overview

The motor encoder vector control is used to control motor rotation in an application. This module is meant to be used with Surface Permanent Magnet (SPM) motors and allows applications to start or stop motor rotation easily.

### Features

The motor encoder module has below features.

- Start/stop motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

### Target Hardware

The below figure shows an example of target hardware of this Motor Encoder Module.

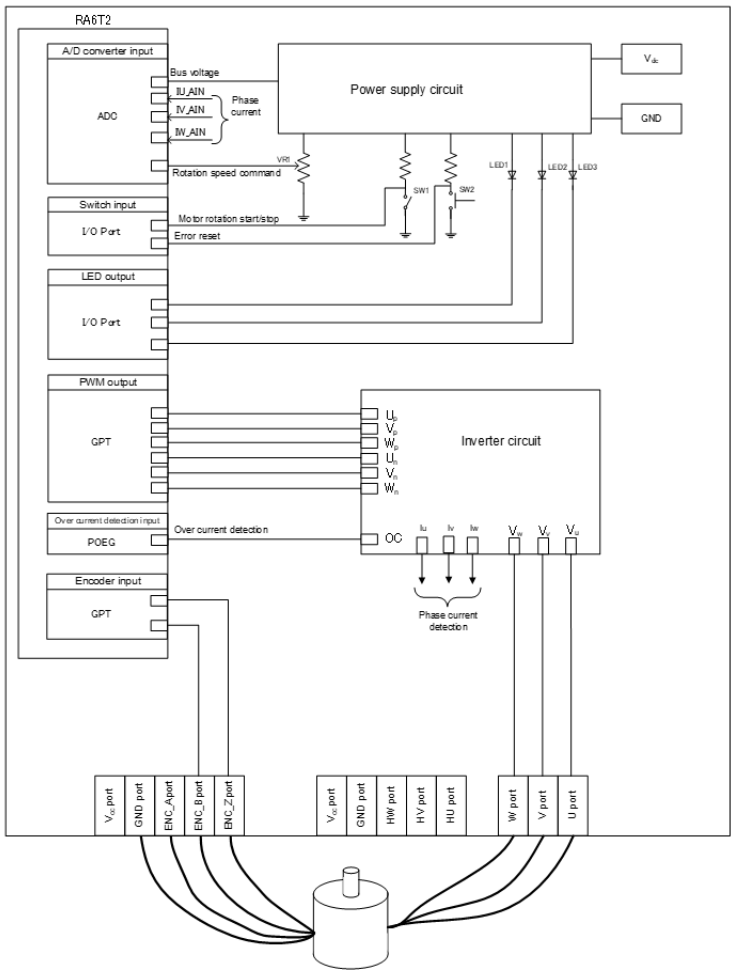



Figure 272: Example of target hardware of motor encoder module

## Block Diagram

The below figure shows block diaram of encoder vector motor control.

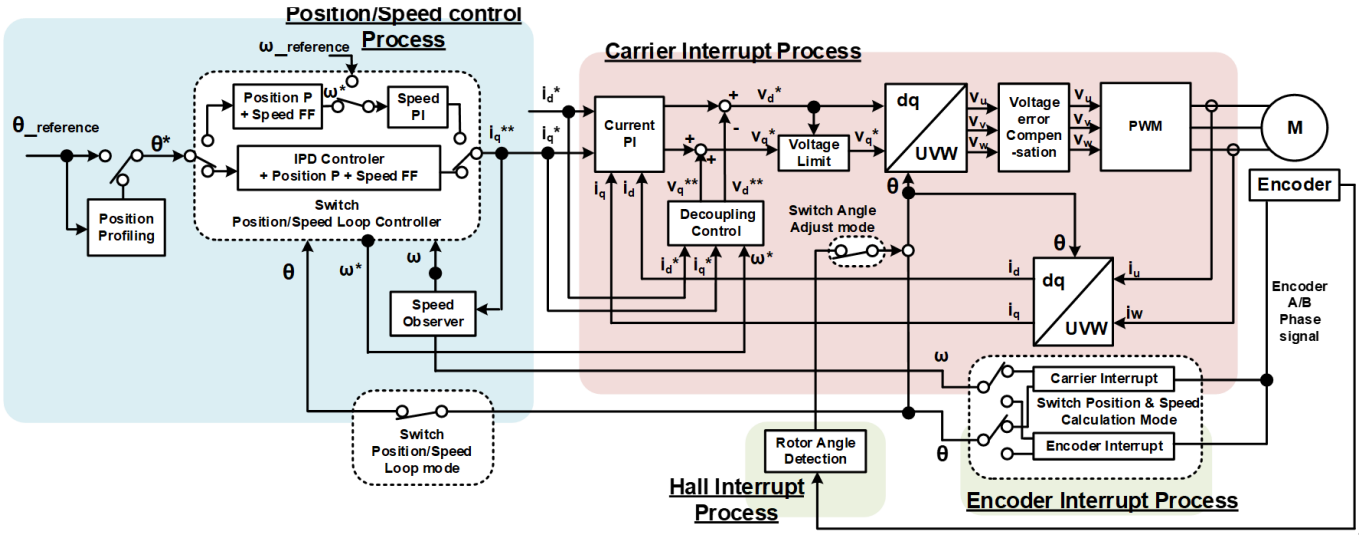



Figure 273: Block diagram of encoder vector control (PI feedback loop)

### Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.

$$m = \frac{V}{E}$$

m : Modulation ratio V : Reference voltage E : Inverter input voltage

Figure 274: Modulation factor

- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

$$\begin{pmatrix} V_u' \\ V_v' \\ V_w' \end{pmatrix} = \begin{pmatrix} V_u \\ V_v \\ V_w \end{pmatrix} + \Delta V \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\because \Delta V = -\frac{V_{max}+V_{min}}{2},\ V_{max} = max\{V_u, V_v, V_w\},\ V_{min} = min\{V_u, V_v, V_w\}$$

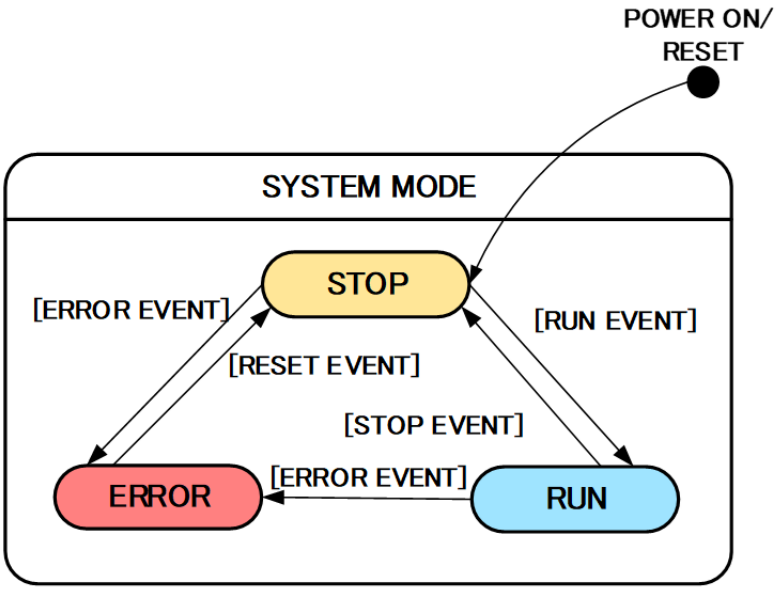
$V_u, V_v, V_w$: Command values of U-, V-, and W-phases

$V_u', V_v', V_w'$: Command values of U-, V-, and W-phases for PWM generation (modulation wave)

Figure 275: Space vector modulation

### State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".




| | | MODE | | |
|---|---|---|---|---|
| | | STOP | RUN | ERROR |
| EVENT | STOP | STOP | STOP | ERROR |
| | RUN | RUN | RUN | ERROR |
| | ERROR | ERROR | ERROR | ERROR |
| | RESET | STOP | RUN | STOP |

Figure 276: State transition diagram

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

| EVENT name | Occurrence factor |
|---|---|
| STOP | by user operation |
| RUN | by user operation |
| ERROR | when the system detects an error |
| RESET | by user operation |

**Flowchart**

The below figures show flowcharts of motor encoder module.

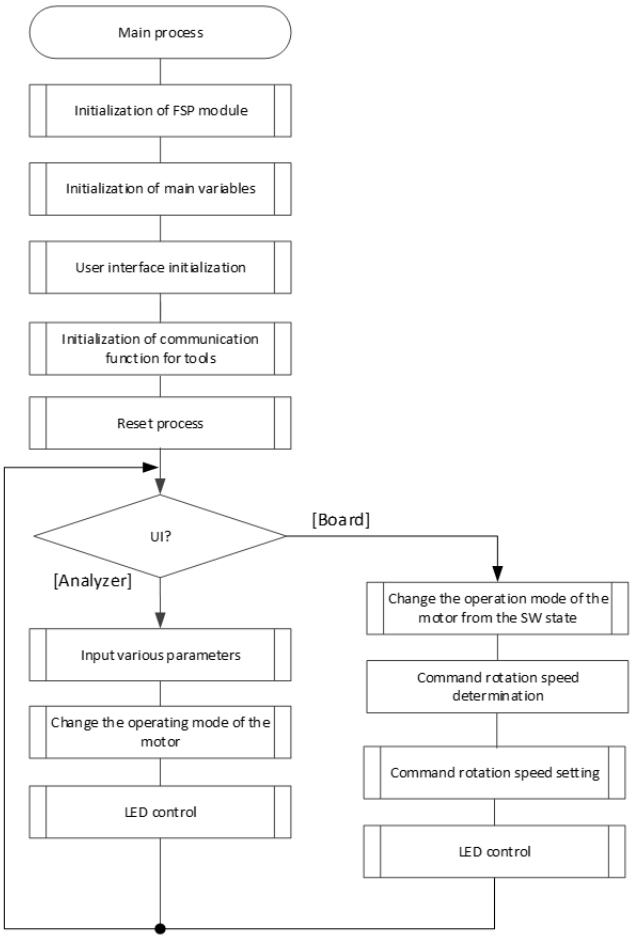



Figure 277: Main process

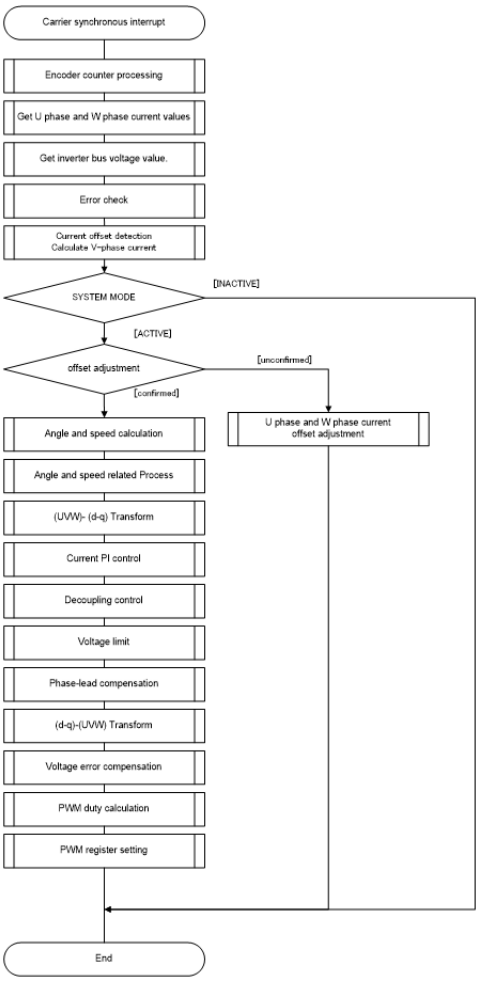



Figure 278: Current control process

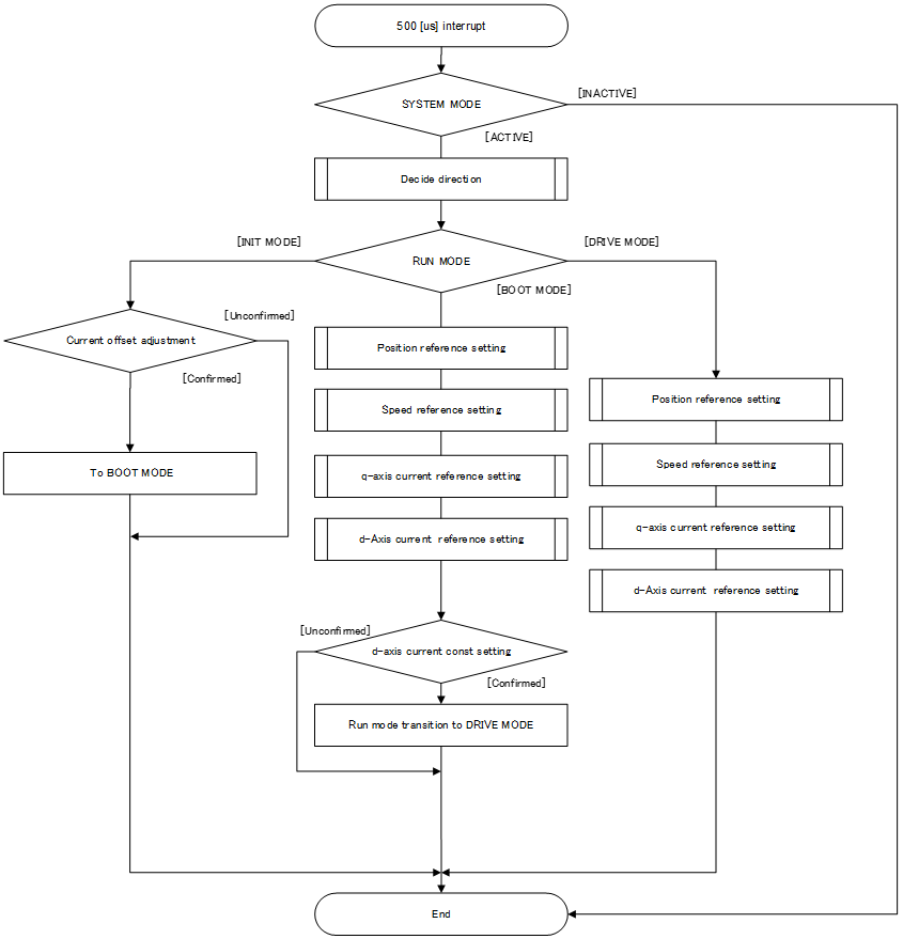



Figure 279: Speed control process

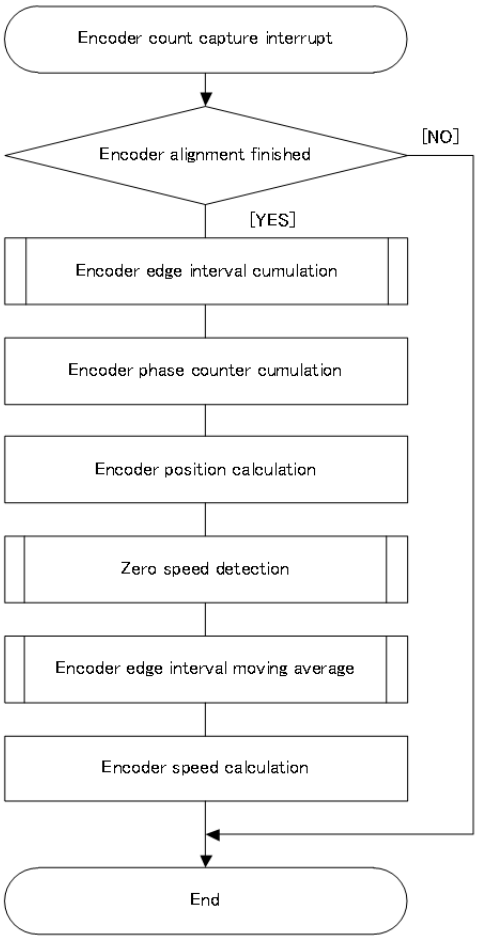



Figure 280: Encoder interrupt process

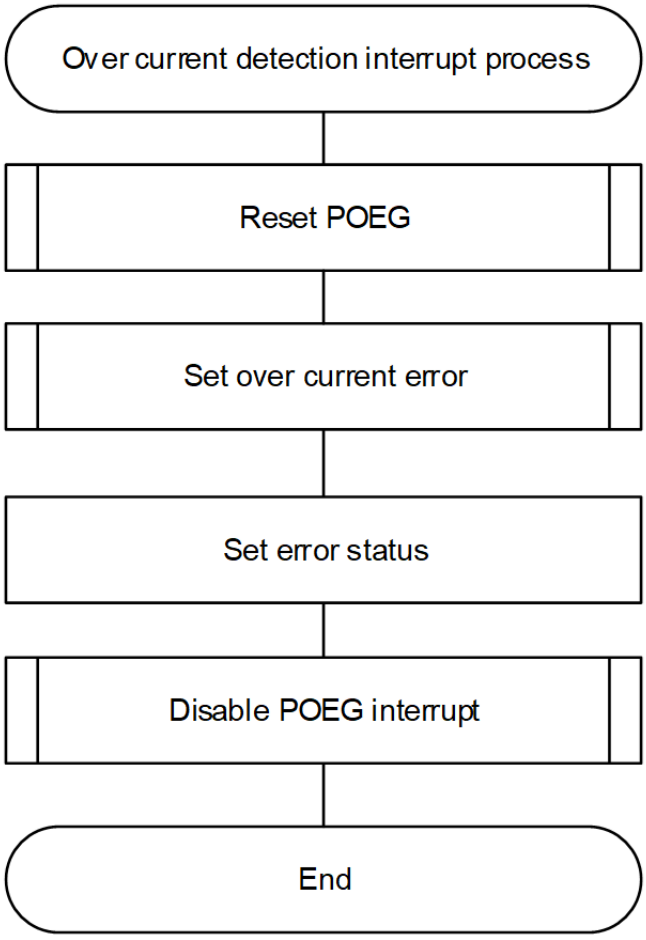



Figure 281: Over current detection interrupt process

# Configuration

### Build Time Configurations for rm_motor_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_encoder_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Encoder Vector Control (rm_motor_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Encoder Vector Control (rm_motor_encoder).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_motor_encoder0 | Module name. |
| Limit of over current (A) | Must be a valid value | 2.0F | Limit of over current.(Detection threshold) |
| Limit of over voltage (V) | Must be a valid value | 28.0F | Limit of over voltage.(Detection threshold) |
| Limit of over speed (rpm) | Must be a valid value | 2100.0F | Limit of over speed.(Detection threshold) |
| Limit of low voltage (V) | Must be a valid value | 18.0F | Limit of low voltage.(Detection threshold) |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt. |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

### Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

## Usage Notes

### Limitations

## Examples

### Basic Example

This is a basic example of minimal use of the motor encoder module in an application.

```
void motor_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
    handle_error(err);
```

```
 /* Set speed reference before motor run */
    (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
 /* Set position reference before motor run */
    (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
 /* Start motor rotation */
    (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_encoder_ctrl_t |
| enum | motor_encoder_ctrl_event_t |

## Enumeration Type Documentation

### ◆ motor_encoder_ctrl_t

| enum motor_encoder_ctrl_t | |
|---|---|
| Enumerator | |
| MOTOR_ENCODER_CTRL_STOP | Stop mode. |
| MOTOR_ENCODER_CTRL_RUN | Run mode. |
| MOTOR_ENCODER_CTRL_ERROR | Error mode. |

### ◆ motor_encoder_ctrl_event_t

| enum motor_encoder_ctrl_event_t | |
|---|---|
| Enumerator | |
| MOTOR_ENCODER_CTRL_EVENT_STOP | Stop event. |
| MOTOR_ENCODER_CTRL_EVENT_RUN | Run event. |
| MOTOR_ENCODER_CTRL_EVENT_ERROR | Error event. |
| MOTOR_ENCODER_CTRL_EVENT_RESET | Reset event. |

## Function Documentation

### ◆ RM_MOTOR_ENCODER_Open()

fsp_err_t RM_MOTOR_ENCODER_Open ( motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg* )

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MOTOR successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

### ◆ RM_MOTOR_ENCODER_Close()

fsp_err_t RM_MOTOR_ENCODER_Close ( motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Encoder Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
   (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_Reset()

fsp_err_t RM_MOTOR_ENCODER_Reset ( motor_ctrl_t *const *p_ctrl*)

Reset Motor Encoder Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
   (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_Run()

fsp_err_t RM_MOTOR_ENCODER_Run ( motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_Stop()

fsp_err_t RM_MOTOR_ENCODER_Stop ( motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_ErrorSet()

fsp_err_t RM_MOTOR_ENCODER_ErrorSet ( motor_ctrl_t *const *p_ctrl*, motor_error_t const *error* )

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_ENCODER_SpeedSet ( motor_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_ENCODER_PositionSet()

fsp_err_t RM_MOTOR_ENCODER_PositionSet ( motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position* )

Set position reference[degree]. Implements motor_api_t::positionSet.

Example:

```
/* Set position reference before motor run */
   (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data set pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_ENCODER_StatusGet()

fsp_err_t RM_MOTOR_ENCODER_StatusGet ( motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status* )

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_ENCODER_AngleGet()

fsp_err_t RM_MOTOR_ENCODER_AngleGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad* )

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_ENCODER_SpeedGet()

fsp_err_t RM_MOTOR_ENCODER_SpeedGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_ENCODER_ErrorCheck()

fsp_err_t RM_MOTOR_ENCODER_ErrorCheck ( motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error* )

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_ENCODER_WaitStopFlagGet()

fsp_err_t RM_MOTOR_ENCODER_WaitStopFlagGet ( motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_ENCODER_FunctionSelect()

fsp_err_t RM_MOTOR_ENCODER_FunctionSelect ( motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function* )

Select using function. Implements motor_api_t::functionSelect.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_ENCODER_InertiaEstimateStart()

fsp_err_t RM_MOTOR_ENCODER_InertiaEstimateStart ( motor_ctrl_t *const *p_ctrl*)

Start inertia estimation function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_ENCODER_InertiaEstimateStop()

fsp_err_t RM_MOTOR_ENCODER_InertiaEstimateStop ( motor_ctrl_t *const *p_ctrl*)

Stop(Cancel) inertia estimation function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_ENCODER_ReturnOriginStart()

fsp_err_t RM_MOTOR_ENCODER_ReturnOriginStart ( motor_ctrl_t *const *p_ctrl*)

Start return origin function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_ENCODER_ReturnOriginStop()

| fsp_err_t RM_MOTOR_ENCODER_ReturnOriginStop ( motor_ctrl_t *const *p_ctrl*) |
|---|
| Stop(Cancel) return origin function.<br><br>**Return values**<br>FSP_SUCCESS — Successfully resetted.<br>FSP_ERR_ASSERTION — Null pointer.<br>FSP_ERR_NOT_OPEN — Module is not open.<br>FSP_ERR_INVALID_MODE — Mode unmatch<br><br>***Note*** |

## 5.2.11.12 Motor Inertia estimate (rm_motor_inertia_estimate)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_Open (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg)<br>Opens and configures the motor inertia estimate module. Implements motor_inertia_estimate_api_t::open. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_Close (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Disables specified motor inertia estimate module. Implements motor_inertia_estimate_api_t::close. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_Start (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Start inertia estimation. Implements motor_inertia_estimate_api_t::start. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_Stop (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Stop (Cancel) inertia estimation. Implements motor_inertia_estimate_api_t::stop. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_Reset (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Reset variables of inertia estimate module. Implements motor_inertia_estimate_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_InfoGet (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info)<br>Get information of inertia estimation. Implements motor_inertia_estimate_api_t::infoGet. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_DataSet (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data)<br>Set necessary data to inertia estimation. Implements motor_inertia_estimate_api_t::dataSet. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Cyclic process of inertia estimation at speed control period. Implements motor_inertia_estimate_api_t::speedCyclic. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic (motor_inertia_estimate_ctrl_t *const p_ctrl)<br>Cyclic process of inertia estimation at current control period (called at A/D conversion finish interrupt). Implements motor_inertia_estimate_api_t::currentCyclic. More... |
| fsp_err_t | RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg)<br>Update the parameters of inertia estimate. Implements motor_inertia_estimate_api_t::parameterUpdate. More... |

### Detailed Description

Measurement and calculation process for the motor control on RA MCUs. This module implements the Motor Inertia Estimate Interface.

# Overview

The motor inertia estimation module is used to measure and calculate rotor inertia in an application.

This module should be used with Renesas Motor Workbench (RMW) basically.

### Features

The Motor Inertia Estimation Module has below features.

- Measurement process work automatically
- Calculate rotor inertia [kgm^2].

# Configuration

### Build Time Configurations for rm_motor_inertia_estimate

The following build time configurations are defined in fsp_cfg/rm_motor_inertia_estimate_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor inertia estimation (rm_motor_inertia_estimate)

This module can be added to the Stacks tab via New Stack > Motor > Motor inertia estimation (rm_motor_inertia_estimate).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_inertia_estimate0 | Module name. |
| Moved position distance to measure (degree) | Must be set -360 to -10 or 10 to 360. | 360 | Moved position distance to measure inertia (degree) |
| Maximum speed (rpm) | Must be set over 60. | 500 | Maximum rotation speed (rpm) |
| Acceleration time | Must be a valid non-negative value. | 0.3 | Acceleration time |
| Motor inertia | Must be a valid non-negative value. | 0.0000041 | Motor inertia |
| Low threshold to judge speed | Must be set 0.1 to 0.5. | 0.1 | Low threshold to judge speed reached |
| High threshold to judge speed | Must be set 0.5 to 0.9, and greater than low threshold | 0.9 | High threshold to judge speed reached |
| Time to wait moving stability (sec) | Must be a valid non-negative value. | 0.8 | Time to wait moving stability |
| Cyclic period of current control (sec) | Must be a valid non-negative value. | 0.00005 | Cyclic period of current control (sec) |

| | | | |
|---|---|---|---|
| Cyclic period of speed control (sec) | Must be a valid non-negative value. | 0.0005 | Cyclic period of speed control (sec) |
| Motor pole pairs | Must be a valid non-negative value. | 4 | Motor pole pairs |
| Motor magnet flux (Wb) | Must be a valid non-negative value. | 0.00623 | Motor magnet flux (Wb) |
| Interval time | Must be a valid non-negative value. | 400.0 | Interval time. Please set same value as Position control |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Limitations

## Examples

### Basic Example

This is a basic example of minimal use of the Motor Inertia Estimation in an application.

```
void motor_inertia_estimate_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Open(&g_mtr_inertia_estimate0_ctrl,
&g_mtr_inertia_estimate0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start process. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Start(&g_mtr_inertia_estimate0_ctrl);
    temp_set_data.f_iq = 1.0F;
    temp_set_data.f_speed_radian_control = 1.04F;
    temp_set_data.s2_position_degree    = 180;
    temp_set_data.u1_position_state      = 0;
 /* Set data to the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_DataSet(&g_mtr_inertia_estimate0_ctrl,
```

```
&temp_set_data);
 /* Get information from the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_InfoGet(&g_mtr_inertia_estimate0_ctrl,
&temp_information);
 /* Stop process. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Stop(&g_mtr_inertia_estimate0_ctrl);
 /* Close the module. */
    err = RM_MOTOR_INERTIA_ESTIMATE_Close(&g_mtr_inertia_estimate0_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | motor_inertia_estimate_extended_cfg_t |
| struct | motor_inertia_estimate_instance_ctrl_t |

## Data Structure Documentation

### ◆ motor_inertia_estimate_extended_cfg_t

| struct motor_inertia_estimate_extended_cfg_t | | |
|---|---|---|
| Extended configurations for motor inertia estimate | | |
| Data Fields | | |
| int16_t | s2_move_degree | Moving position reference [degree]. |
| uint16_t | u2_J_max_speed_rpm | Maximum Speed [rpm]. |
| float | f_accel_time | Acceleration time. |
| float | f_rotor_inertia | Initialized rotor inertia value. |
| float | f_judge_low_threshold | Low threshold to judge speed. |
| float | f_judge_high_threshold | High threshold to judge speed. |
| float | f_change_mode_time | Timing value to change internal mode. |
| float | f_current_ctrl_period | Period of current control [sec]. |
| float | f_speed_ctrl_period | Period of speed control [sec]. |
| uint8_t | u1_motor_polepairs | Motor pole pairs. |
| float | f_motor_m | Motor magnet flux [Wb]. |
| float | f_position_interval | Interval counts for reference position change. |

### ◆ motor_inertia_estimate_instance_ctrl_t

| struct motor_inertia_estimate_instance_ctrl_t | | |
|---|---|---|
| Inertia estimate instance control block | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the module is configured. |
| motor_inertia_estimate_start_flag_t | start_flag | start/stop flag |
| motor_inertia_estimate_mode_t | mode | Internal mode. |
| uint8_t | u1_mode_count | Use to manage internal mode. |
| motor_inertia_estimate_period_t | speed_period | Measure period. |
| motor_inertia_estimate_period_t | speed_period_buffer | Buffer of measure period to be referred by current cyclic. |
| uint32_t | u4_measure_count | Counter for speed control cycle. |
| uint32_t | u4_wait_count | Counter to wait change mode timing. |
| uint8_t | u1_position_move_mode | Position move mode (TRIANGLE/TRAPEZOID) |
| int16_t | s2_initial_position_degree | Initial position. |
| float | f_iq_ad | q-axis current [A] |
| float | f_summary_iq_ad | Summary of q-axis current. |
| float | f_position_mode_time | Summary of speed control period to judge the timing. |
| float | f_position_dt_time_sec | Differential time of move. |
| int16_t | s2_position_reference_degree | Position reference [degree]. |
| float | f_estimated_value | Estimated inertia. |
| float | f_inertia_value1 | Buffer to calculate inertia 1. |
| float | f_inertia_value2 | Buffer to calculate inertia 2. |
| float | f_interval_time | Interval time about position transition. |
| float | f_inertia_speed_ctrl1 | |
| float | f_inertia_speed_ctrl2 | |
| float | f_inertia_speed_ctrl3 | |
| float | f_inertia_speed_ctrl4 | |
| float | f_inertia_speed_ctrl5 | |
| float | f_inertia_speed_ctrl6 | |
| float | f_inertia_speed_ctrl7 | |
| | | |

| | | |
|---|---|---|
| float | f_inertia_speed_ctrl8 | |
| float | f_inertia_integ_iq1 | |
| float | f_inertia_integ_iq2 | |
| float | f_inertia_integ_iq3 | |
| float | f_inertia_integ_iq4 | |
| float | f_inertia_integ_time1 | |
| float | f_inertia_integ_time2 | |
| float | f_inertia_integ_time3 | |
| float | f_inertia_integ_time4 | |
| float | f_inverse_motor_polepairs | Inverse motor pole pairs (for calculation) |
| motor_inertia_estimate_set_data_t | receive_data | Received data set from speed(position) and current. |
| motor_inertia_estimate_cfg_t const * | p_cfg | Pointer of configuration structure. |

### Function Documentation

#### ◆ RM_MOTOR_INERTIA_ESTIMATE_Open()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Open ( motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_cfg_t const *const *p_cfg* )

Opens and configures the motor inertia estimate module. Implements motor_inertia_estimate_api_t::open.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Open(&g_mtr_inertia_estimate0_ctrl,
&g_mtr_inertia_estimate0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor inertia estimate module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_Close()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Close ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Disables specified motor inertia estimate module. Implements motor_inertia_estimate_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Close(&g_mtr_inertia_estimate0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_Start()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Start ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Start inertia estimation. Implements motor_inertia_estimate_api_t::start.

Example:

```
/* Start process. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Start(&g_mtr_inertia_estimate0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_Stop()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Stop ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Stop (Cancel) inertia estimation. Implements motor_inertia_estimate_api_t::stop.

Example:

```
/* Stop process. */
   err = RM_MOTOR_INERTIA_ESTIMATE_Stop(&g_mtr_inertia_estimate0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully stopped (canceled). |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_Reset()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_Reset ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Reset variables of inertia estimate module. Implements motor_inertia_estimate_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_InfoGet()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_InfoGet ( motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_info_t *const *p_info* )

Get information of inertia estimation. Implements motor_inertia_estimate_api_t::infoGet.

Example:

```
/* Get information from the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_InfoGet(&g_mtr_inertia_estimate0_ctrl,
&temp_information);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully get data. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Argument pointer is invalid. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_DataSet()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_DataSet ( motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_set_data_t *const *p_set_data* )

Set necessary data to inertia estimation. Implements motor_inertia_estimate_api_t::dataSet.

Example:

```
/* Set data to the module. */
   err = RM_MOTOR_INERTIA_ESTIMATE_DataSet(&g_mtr_inertia_estimate0_ctrl,
&temp_set_data);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set data. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_SpeedCyclic ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Cyclic process of inertia estimation at speed control period. Implements motor_inertia_estimate_api_t::speedCyclic.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully perform the process. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_CurrentCyclic ( motor_inertia_estimate_ctrl_t *const *p_ctrl*)

Cyclic process of inertia estimation at current control period (called at A/D conversion finish interrupt). Implements motor_inertia_estimate_api_t::currentCyclic.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully perform the process. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate()

fsp_err_t RM_MOTOR_INERTIA_ESTIMATE_ParameterUpdate ( motor_inertia_estimate_ctrl_t *const *p_ctrl*, motor_inertia_estimate_cfg_t const *const *p_cfg* )

Update the parameters of inertia estimate. Implements motor_inertia_estimate_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### 5.2.11.13 Motor Position Controller (rm_motor_position)

Modules » Motor

#### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_POSITION_Open (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)<br>Opens and configures the Motor Position Module. Implements motor_position_api_t::open. More... |
| fsp_err_t | RM_MOTOR_POSITION_Close (motor_position_ctrl_t *const p_ctrl)<br>Disables specified Motor Position Module. Implements motor_position_api_t::close. More... |
| fsp_err_t | RM_MOTOR_POSITION_Reset (motor_position_ctrl_t *const p_ctrl)<br>Reset the variables of Motor Position Module. Implements motor_position_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_POSITION_PositionGet (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)<br>Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet. More... |
| fsp_err_t | RM_MOTOR_POSITION_PositionSet (motor_position_ctrl_t *const p_ctrl, float const position_rad)<br>Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_POSITION_PositionReferenceSet (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)<br>Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet. More... |
| fsp_err_t | RM_MOTOR_POSITION_ControlModeSet (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)<br>Set Position Control Mode. Implements motor_position_api_t::controlModeSet. More... |
| fsp_err_t | RM_MOTOR_POSITION_PositionControl (motor_position_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl. More... |
| fsp_err_t | RM_MOTOR_POSITION_IpdSpeedPControl (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)<br>Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl. More... |
| fsp_err_t | RM_MOTOR_POSITION_SpeedReferencePControlGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)<br>Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet. More... |
| fsp_err_t | RM_MOTOR_POSITION_SpeedReferenceIpdControlGet (motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)<br>Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet. More... |
| fsp_err_t | RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)<br>Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet. More... |
| fsp_err_t | RM_MOTOR_POSITION_InfoGet (motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info)<br>Get position information. More... |
| fsp_err_t | RM_MOTOR_POSITION_ParameterUpdate (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)<br>Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor position Interface.

# Overview

The motor position is used to control the position of motor rotor in an application. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module calculates speed reference with inputted position reference and current rotational speed.

**Features**

The Motor Position Module has below features.

- Calculate speed reference.

# Configuration

**Build Time Configurations for rm_motor_position**

The following build time configurations are defined in fsp_cfg/rm_motor_position_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Motor > Motor Position Controller (rm_motor_position)**

This module can be added to the Stacks tab via New Stack > Motor > Motor Position Controller (rm_motor_position).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_position0 | Module name. |
| Position dead band | Must be a valid non-negative value. | 1U | Ignored area of position control. |
| Position band limit | Must be a valid non-negative value. | 3U | Ignored area of position control. |
| Speed feedforward ratio | Must be a valid value | 0.8F | Speed feedforward ratio. |
| Encoder counts per one rotation | Must be a valid value | 1200.0F | Encoder counts per one rotation. |
| Position omega | Must be a valid value | 10.0F | Position control omega. |
| Period of speed control (sec) | Must be a valid value | 0.0005F | Period of speed control. |
| IPD | | | |
| IPD LPF | • Disable<br>• Enable | Disable | IPD LPF process enable or disable |
| Position Kp ratio | Must be a valid value | 0.3F | Position Kp ratio. |
| Position feedforward | Must be a valid value | 0.0F | Position feedforward |

| | | | |
|---|---|---|---|
| ratio | | | ratio. |
| Speed K ratio | Must be a valid value | 2.0F | Speed K ratio |
| Error Limit #1 | Must be a valid value | 10.0F | Error limitation #1 |
| Error limit #2 | Must be a valid value | 0.2F | Error limitation #2 |
| LPF omega | Must be a valid value | 500.0F | LPF omega. |
| LPF zeta | Must be a valid value | 1.0F | LPF zeta. |
| Position Profiling | | | |
| Interval time | Must be a valid non-negative value. | 400U | Interval time. |
| Accel time | Must be a valid value | 0.3F | Accel time. |
| Maximum accel time (sec) | Must be a valid value | 8117.96F | Maximum acceleration time (sec) |
| Acceleration maximum speed | Must be a valid value | 2000.0F | Acceleration maximum speed. |
| Update step of timer | Must be a valid value | 0.0005F | Update step of timer. |
| Motor Parameter | | | |
| Pole pair | Value must be non-negative | 7 | Pole pair |
| Resistance (ohm) | Must be a valid value | 0.453F | Resistance |
| Inductance of d-axis (H) | Must be a valid value | 0.0009447F | Inductance of d-axis |
| Inductance of q-axis (H) | Must be a valid value | 0.0009447F | Inductance of q-axis |
| Permanent magnetic flux (Wb) | Must be a valid value | 0.006198F | Permanent magnetic flux |
| Motor Parameter > Motor inertia (kgm^2) | Must be a valid value | 0.00000962F | Motor inertia |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Limitations

- Set the period of speed control with non-negative value.
- Set the limit of speed change step with non-negative value.
- Set the maximum speed with non-negative value.

# Examples

### Basic Example

This is a basic example of using the Motor Position module in an application.

```
void motor_position_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_POSITION_Open(g_motor_position0.p_ctrl, g_motor_position0.p_cfg);
    handle_error(err);
 /* Set working mode */
    RM_MOTOR_POSITION_ModeSet(g_motor_position0.p_ctrl,
MOTOR_POSITION_CTRL_MODE_STEP);
 /* Set position reference */
 RM_MOTOR_POSITION_PositionReferenceSet(g_motor_position0.p_ctrl, 180U);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Perform Position Control Process */
 RM_MOTOR_POSITION_PositionControl(g_motor_position0.p_ctrl);
 /* Perform Speed IPD Control Process */
 RM_MOTOR_POSITION_IpdSpeedPControl(g_motor_position0.p_ctrl, 0.0F, 0.0F,
&temp_iq_ref);
 /* Get Position */
 RM_MOTOR_POSITION_PositionGet(g_motor_position0.p_ctrl, &temp_position);
 /* Update parameters */
 RM_MOTOR_POSITION_ParameterUpdate(g_motor_position0.p_ctrl,
&g_motor_position0.p_cfg);
    }
 /* Reset Speed Control */
 RM_MOTOR_POSITION_Reset(g_motor_position0.p_ctrl);
 /* Close Speed Control */
```

```
 RM_MOTOR_POSITION_Close(g_motor_position0.p_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_position_ipd_lpf_t |

## Enumeration Type Documentation

### ◆ motor_position_ipd_lpf_t

enum motor_position_ipd_lpf_t

| Enumerator | |
|---|---|
| MOTOR_POSITION_IPD_LPF_DISABLE | ipd control is disabled |
| MOTOR_POSITION_IPD_LPF_ENABLE | ipd control is enabled |

## Function Documentation

### ◆ RM_MOTOR_POSITION_Open()

fsp_err_t RM_MOTOR_POSITION_Open ( motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg* )

Opens and configures the Motor Position Module. Implements motor_position_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor Position Module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Set parameter is invalid. |

### ◆ RM_MOTOR_POSITION_Close()

fsp_err_t RM_MOTOR_POSITION_Close ( motor_position_ctrl_t *const *p_ctrl*)

Disables specified Motor Position Module. Implements motor_position_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_Reset()

fsp_err_t RM_MOTOR_POSITION_Reset ( motor_position_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Position Module. Implements motor_position_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_PositionGet()

fsp_err_t RM_MOTOR_POSITION_PositionGet ( motor_position_ctrl_t *const *p_ctrl*, int16_t *const *p_position* )

Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_PositionSet()

fsp_err_t RM_MOTOR_POSITION_PositionSet ( motor_position_ctrl_t *const *p_ctrl*, float const *position_rad* )

Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_PositionReferenceSet()

fsp_err_t RM_MOTOR_POSITION_PositionReferenceSet ( motor_position_ctrl_t *const *p_ctrl*, int16_t const *position_reference_deg* )

Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_ControlModeSet()

fsp_err_t RM_MOTOR_POSITION_ControlModeSet ( motor_position_ctrl_t *const *p_ctrl*, motor_position_ctrl_mode_t const *mode* )

Set Position Control Mode. Implements motor_position_api_t::controlModeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_PositionControl()

fsp_err_t RM_MOTOR_POSITION_PositionControl ( motor_position_ctrl_t *const *p_ctrl*)

Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data calculation. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_POSITION_IpdSpeedPControl()

fsp_err_t RM_MOTOR_POSITION_IpdSpeedPControl ( motor_position_ctrl_t *const *p_ctrl*, float const *ref_speed_rad*, float const *speed_rad*, float *const *p_iq_ref* )

Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data calculation. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_SpeedReferencePControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferencePControlGet ( motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref* )

Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_SpeedReferenceIpdControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceIpdControlGet ( motor_position_ctrl_t *const *p_ctrl*, float const *max_speed_rad*, float *const *p_speed_ref* )

Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet ( motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref* )

Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_InfoGet()

fsp_err_t RM_MOTOR_POSITION_InfoGet ( motor_position_ctrl_t *const *p_ctrl*, motor_position_info_t *const *p_info* )

Get position information.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_POSITION_ParameterUpdate()

| fsp_err_t RM_MOTOR_POSITION_ParameterUpdate ( motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg* ) |
|---|
| Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate.<br><br>**Return values**<br>FSP_SUCCESS — Successfully data was updated.<br>FSP_ERR_ASSERTION — Null pointer.<br>FSP_ERR_NOT_OPEN — Module is not open.<br>FSP_ERR_INVALID_ARGUMENT — Configuration parameter error. |

## 5.2.11.14 Motor Sensorless Vector Control (rm_motor_sensorless)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SENSORLESS_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MOTOR_SENSORLESS_Close (motor_ctrl_t *const p_ctrl)<br>Disables specified Motor Sensorless Control block. Implements motor_api_t::close. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_Reset (motor_ctrl_t *const p_ctrl)<br>Reset Motor Sensorless Control block. Implements motor_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_Run (motor_ctrl_t *const p_ctrl)<br>Run Motor (Start motor rotation). Implements motor_api_t::run. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_Stop (motor_ctrl_t *const p_ctrl)<br>Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_ErrorSet (motor_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | motor_error_t const error)<br>Set error information. Implements motor_api_t::errorSet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed reference[rpm]. Implements motor_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)<br>Get current control status. Implements motor_api_t::statusGet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)<br>Get current rotor angle. Implements motor_api_t::angleGet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get rotational speed. Implements motor_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)<br>Check the occurunce of Error. Implements motor_api_t::errorCheck. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)<br>Set position reference. Implements motor_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)<br>Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_SENSORLESS_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)<br>Select function. Implements motor_api_t::functionSelect. More... |

### Detailed Description

Usual control of a SPM motor on RA MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless).

# Overview

The motor sensorless vector control is used to control a motor rotation in an application. This module is implemented with using SPM motor. User can start/stop motor rotation simply.

### Features

The Motor Sensorless Module has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

### Target Hardware

The below figure shows an example of target hardware of this Motor Sensorless Module.

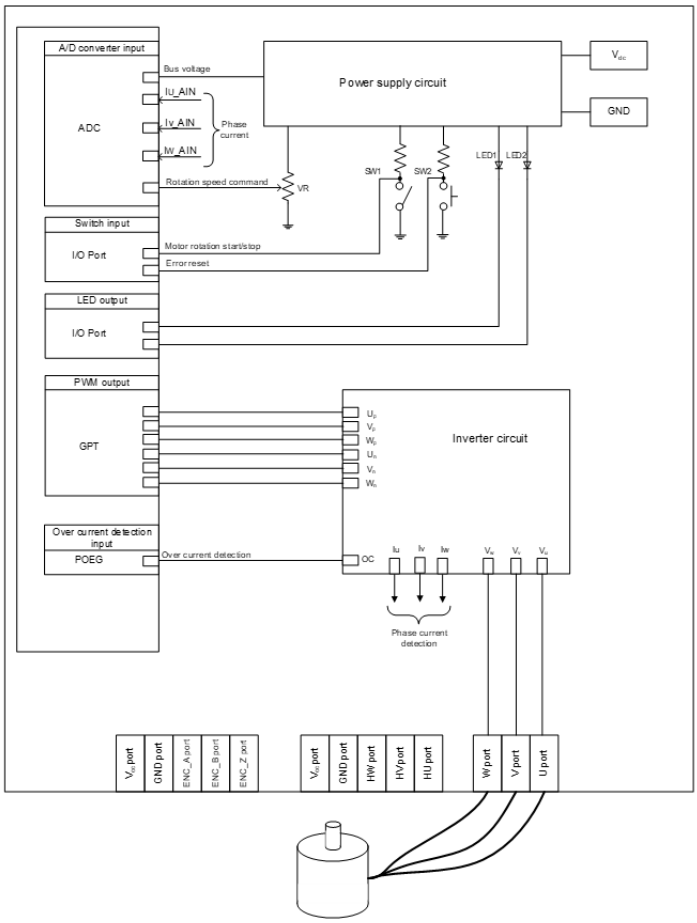



Figure 282: Example of target hardware of motor sensorless module

### Block Diagram

The below figures show block diaram of sensorless vector motor control. The 1st shows as an open-loop state, 2nd as a PI feedback loop state.

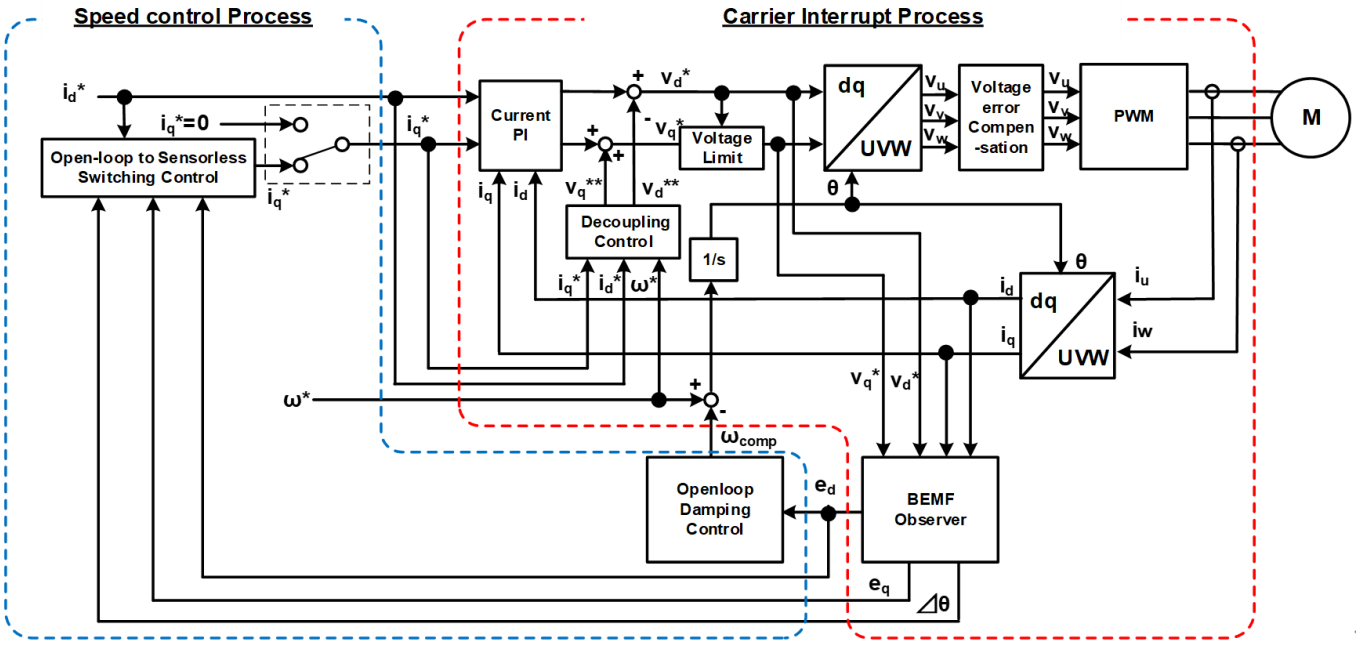



Figure 283: Block diagram of sensorless vector control (open-loop)

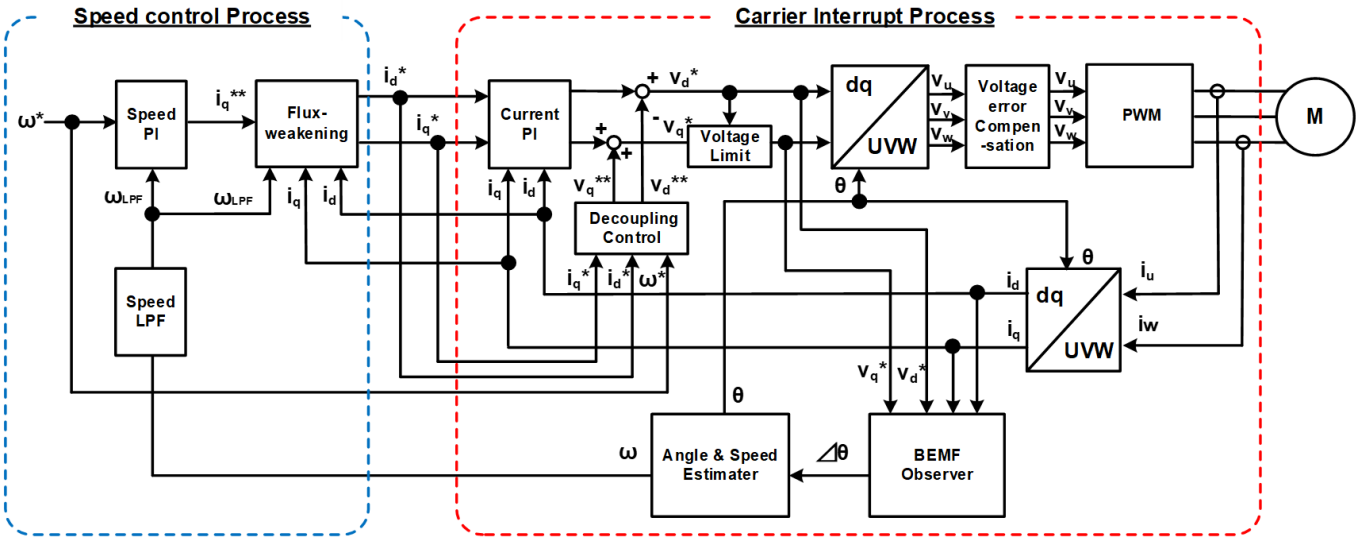



Figure 284: Block diagram of sensorless vector control (PI feedback loop)

## Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.

$$m = \frac{V}{E}$$

m : Modulation ratio V : Reference voltage E : Inverter input voltage

Figure 285: Modulation factor

- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

$$\begin{pmatrix} V_u' \\ V_v' \\ V_w' \end{pmatrix} = \begin{pmatrix} V_u \\ V_v \\ V_w \end{pmatrix} + \Delta V \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\because \Delta V = -\frac{V_{max}+V_{min}}{2},\ V_{max} = max\{V_u, V_v, V_w\},\ V_{min} = min\{V_u, V_v, V_w\}$$

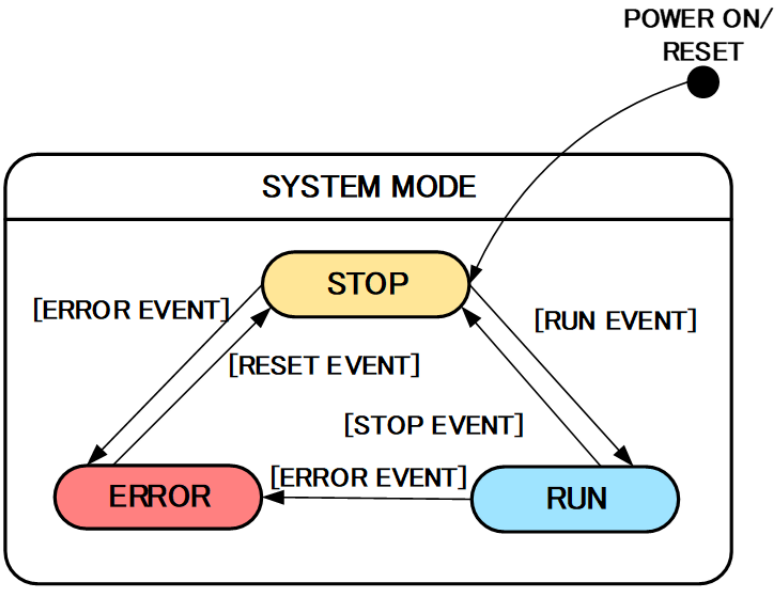
$V_u, V_v, V_w$: Command values of U-, V-, and W-phases

$V_u', V_v', V_w'$: Command values of U-, V-, and W-phases for PWM generation (modulation wave)

Figure 286: Space vector modulation

### State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".




| | | MODE | | |
|---|---|---|---|---|
| | | STOP | RUN | ERROR |
| EVENT | STOP | STOP | STOP | ERROR |
| | RUN | RUN | RUN | ERROR |
| | ERROR | ERROR | ERROR | ERROR |
| | RESET | STOP | RUN | STOP |

Figure 287: State transition diagram

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

| EVENT name | Occurrence factor |
|---|---|
| STOP | by user operation |
| RUN | by user operation |
| ERROR | when the system detects an error |
| RESET | by user operation |

**Flowchart**

The below figures show flowcharts of motor sensorless module.

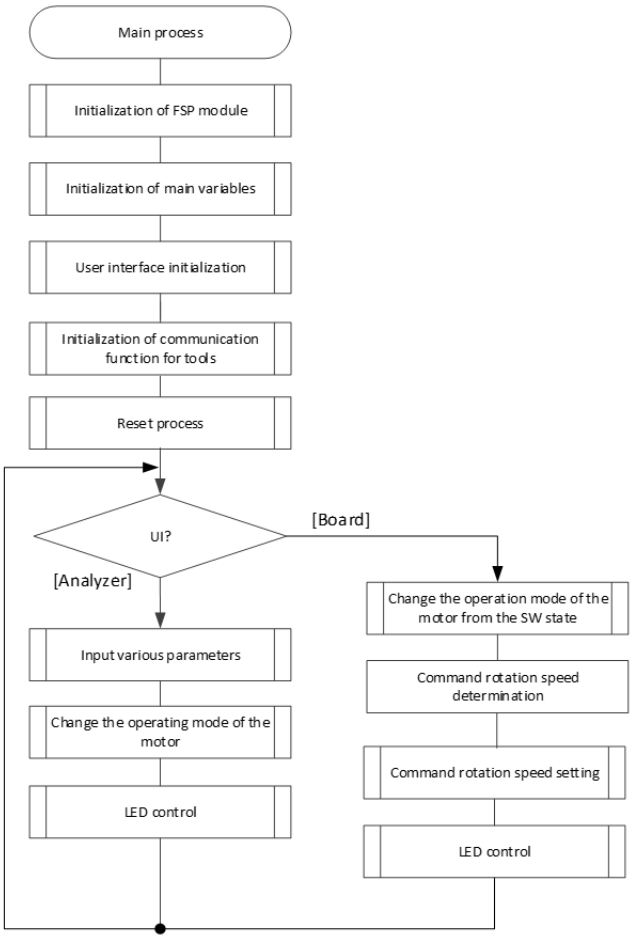



Figure 288: Main process

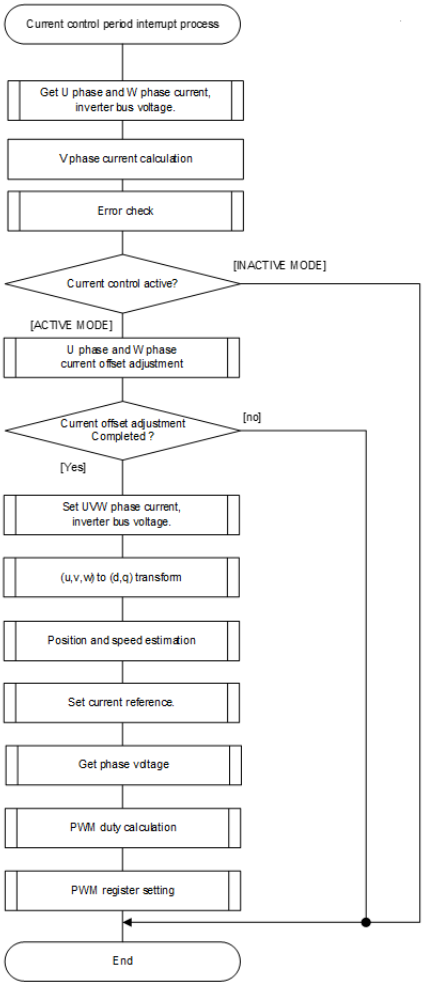



Figure 289: Current control process

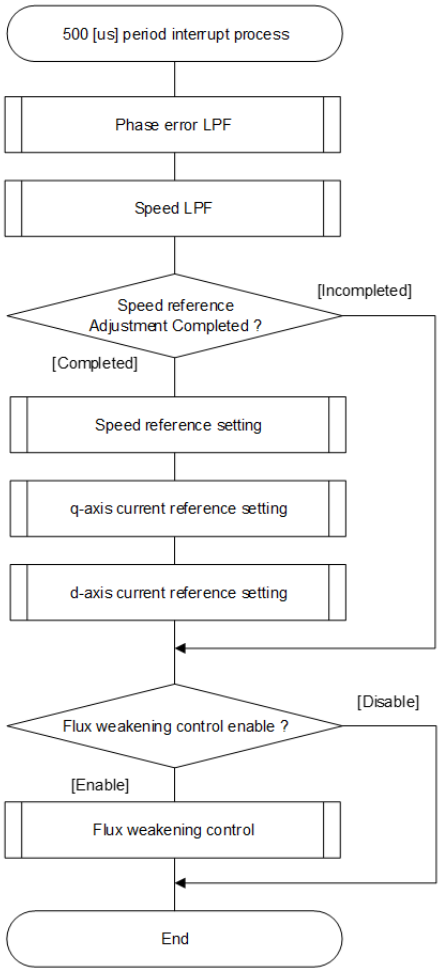



Figure 290: Speed control process

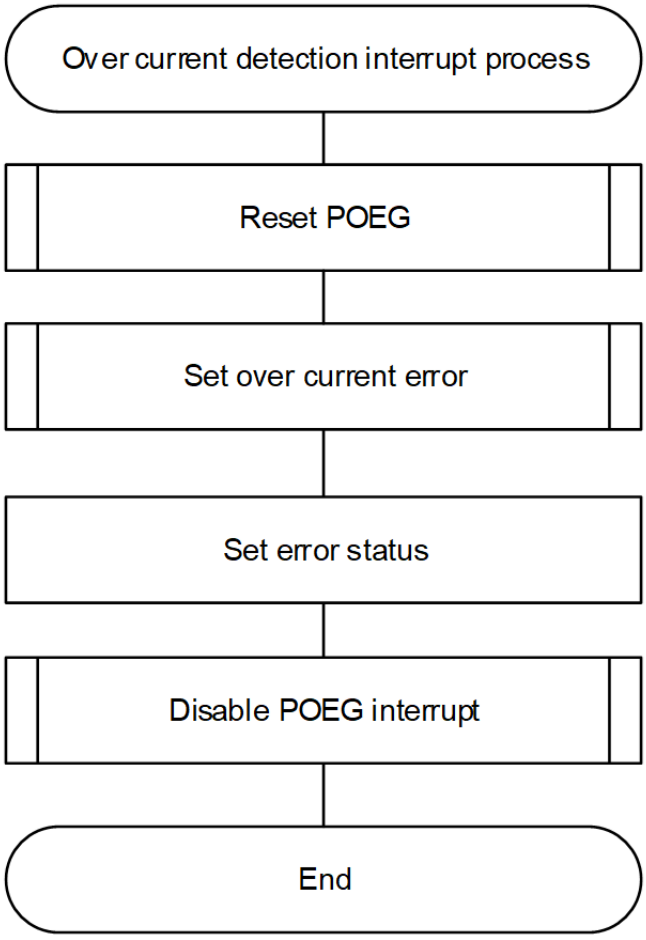



Figure 291: Over current detection interrupt process

# Configuration

### Build Time Configurations for rm_motor_sensorless

The following build time configurations are defined in fsp_cfg/rm_motor_sensorless_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Sensorless Vector Control (rm_motor_sensorless)

This module can be added to the Stacks tab via New Stack > Motor > Motor Sensorless Vector Control (rm_motor_sensorless).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_motor_sensorless0 | Module name. |
| Limit of over current (A) | Must be a valid value | 0.42F | Limit of over current.(Detection threshold) |
| Limit of over voltage (V) | Must be a valid value | 28.0F | Limit of over voltage.(Detection threshold) |
| Limit of over speed (rpm) | Must be a valid value | 3000.0F | Limit of over speed.(Detection threshold) |
| Limit of low voltage (V) | Must be a valid value | 14.0F | Limit of low voltage.(Detection threshold) |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt. |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

### Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

## Usage Notes

### Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

## Examples

### Basic Example

This is a basic example of minimal use of the Motor Sensorless in an application.

```
void motor_sensorless_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
 /* Start motor rotation */
    (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
 /* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
}
```

### Function Documentation

### ◆ RM_MOTOR_SENSORLESS_Open()

fsp_err_t RM_MOTOR_SENSORLESS_Open ( motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg* )

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MOTOR successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_Close()

fsp_err_t RM_MOTOR_SENSORLESS_Close ( motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Sensorless Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
   (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_Reset()

fsp_err_t RM_MOTOR_SENSORLESS_Reset ( motor_ctrl_t *const *p_ctrl*)

Reset Motor Sensorless Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
   (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_Run()

fsp_err_t RM_MOTOR_SENSORLESS_Run ( motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_Stop()

fsp_err_t RM_MOTOR_SENSORLESS_Stop ( motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_ErrorSet()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorSet ( motor_ctrl_t *const *p_ctrl*, motor_error_t const *error* )

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_SpeedSet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedSet ( motor_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_StatusGet()

fsp_err_t RM_MOTOR_SENSORLESS_StatusGet ( motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status* )

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_AngleGet()

fsp_err_t RM_MOTOR_SENSORLESS_AngleGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad* )

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_SpeedGet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_ErrorCheck()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorCheck ( motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error* )

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_PositionSet()

fsp_err_t RM_MOTOR_SENSORLESS_PositionSet ( motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position* )

Set position reference. Implements motor_api_t::positionSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_WaitStopFlagGet()

fsp_err_t RM_MOTOR_SENSORLESS_WaitStopFlagGet ( motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_SENSORLESS_FunctionSelect()

| fsp_err_t RM_MOTOR_SENSORLESS_FunctionSelect ( motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function* ) |
|---|
| Select function. Implements motor_api_t::functionSelect.<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Unsupported.<br><br>***Note*** |

## 5.2.11.15 Motor Speed Controller (rm_motor_speed)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SPEED_Open (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)<br>Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open. More... |
| fsp_err_t | RM_MOTOR_SPEED_Close (motor_speed_ctrl_t *const p_ctrl)<br>Disables specified Motor Speed Module. Implements motor_speed_api_t::close. More... |
| fsp_err_t | RM_MOTOR_SPEED_Reset (motor_speed_ctrl_t *const p_ctrl)<br>Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_SPEED_Run (motor_speed_ctrl_t *const p_ctrl)<br>Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run. More... |
| fsp_err_t | RM_MOTOR_SPEED_SpeedReferenceSet (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)<br>Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet. More... |

| | |
|---|---|
| fsp_err_t | RM_MOTOR_SPEED_PositionReferenceSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)<br>Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet. More... |
| fsp_err_t | RM_MOTOR_SPEED_ParameterSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)<br>Set Input parameters. Implements motor_speed_api_t::parameterSet. More... |
| fsp_err_t | RM_MOTOR_SPEED_SpeedControl (motor_speed_ctrl_t *const p_ctrl)<br>Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl. More... |
| fsp_err_t | RM_MOTOR_SPEED_ParameterGet (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output)<br>Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet. More... |
| fsp_err_t | RM_MOTOR_SPEED_ParameterUpdate (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)<br>Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate. More... |

### Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor speed Interface.

# Overview

The motor speed is used to control the speed of motor rotation in an application. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module caliculates d/q-axis current reference with input speed reference, current rotational speed, and d/q-axis current.

### Features

The motor speed module has below features.

- Calculate d/q-axis electric current reference.
- Flux weakening process at high speed rotation.
- Open loop damping control when using sensorless type.
- Low pass filter of input rotational speed.
- Speed observer function when using encoder type.

# Configuration

### Build Time Configurations for rm_motor_speed

The following build time configurations are defined in fsp_cfg/rm_motor_speed_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Position Support | • Enabled<br>• Disabled | Disabled | Enable position algorithm support. |

### Configurations for Motor > Motor Speed Controller (rm_motor_speed)

This module can be added to the Stacks tab via New Stack > Motor > Motor Speed Controller (rm_motor_speed).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_speed0 | Module name. |
| Speed control period (sec) | Must be a valid non-negative value | 0.0005F | Period of speed control function. |
| Step of speed climbing (rpm) | Must be a valid non-negative value | 0.5F | Step of speed change at start of open-loop. |
| Maximum rotational speed (rpm) | Must be a valid non-negative value. | 2650.0F | Maximum rotational speed (Limit speed). |
| Speed LPF omega | Must be a valid non-negative value | 10.0F | Design parameter for speed LPF. |
| Limit of q-axis current (A) | Must be a valid non-negative value | 0.42F | Limit of q-axis current. |
| Step of speed feedback at open-loop | Must be a valid non-negative value | 0.20F | Step of speed feedback at open-loop. |
| Natural frequency | Must be a valid non-negative value. | 100.0F | Natural frequency for disturbance speed observer. |
| Open-loop damping | • Disable<br>• Enable | Enable | Select enable/disable Open-loop damping control. |
| Flux weakening | • Disable<br>• Enable | Disable | Select enable/disable flux weakening control. |
| Torque compensation for sensorless transition | • Disable<br>• Enable | Enable | Select enable/disable torque compensation for sensorless |

| | | | |
|---|---|---|---|
| | | | transition. |
| Speed observer | • Disable<br>• Enable | Enable | Select enable/disable Speed observer process. |
| Selection of speed observer | • Normal<br>• Disturbance | Normal | Select speed observer type. |
| Control method | • PID<br>• IPD | PID | Select the control method [PID or IPD]. |
| Control type | • Sensoreless<br>• Encoder<br>• Induction<br>• Hall | Sensoreless | |
| Open-Loop | | | |
| Step of d-axis current climbing | Must be a valid non-negative value | 0.3F | Step of d-axis current climbing |
| Step of d-axis current descending | Must be a valid non-negative value | 0.3F | Step of d-axis current descending |
| Step of q-axis current descending ratio | Must be a valid non-negative value | 1.0F | Step of q-axis current descending ratio |
| Reference of d-axis current | Must be a valid non-negative value | 0.3F | Reference of d-axis current |
| Threshold of speed control descending | Must be a valid value | 600.0F | When rotational speed reaches this speed, d-axis current is controlled descending. |
| Threshold of speed control climbing | Must be a valid value | 500.0F | Until rotational speed reaches this speed, d-axis current is controlled climbing. |
| Period between open-loop to BEMF (sec) | Must be a valid non-negative value | 0.025F | Margin time between open-loop control changes to BEMF PI control. |
| Phase error(degree) to decide sensor-less switch timing | Must be a valid value | 10.0F | Phase error(degree) to decide sensor-less switch timing. |
| Design parameter | | | |
| Speed PI loop omega | Must be a valid non-negative value | 5.0F | Speed PI loop omega |
| Speed PI loop zeta | Must be a valid non-negative value | 1.0F | Speed PI loop zeta |
| Estimated d-axis HPF omega | Must be a valid non-negative value | 2.5F | HPF cutoff frequency for ed (Hz) |
| Open-loop damping | Must be a valid non- | 1.0F | Damping ratio of open- |

| | | | |
|---|---|---|---|
| zeta | negative value | | loop damping control |
| Cutoff frequency of phase error LPF | Must be a valid non-negative value | 10.0F | Cutoff frequency of phase error LPF |
| Speed observer omega | Must be a valid non-negative value | 200.0F | Speed observer loop omega |
| Speed observer zeta | Must be a valid non-negative value | 1.0F | Speed observer loop zeta |
| Motor Parameter | | | |
| Pole pairs | Must be a valid integer. | 2 | Pole pairs |
| Resistance (ohm) | Must be a valid non-negative value | 8.5F | Resistance |
| Inductance of d-axis (H) | Must be a valid non-negative value | 0.0045F | Inductance of d-axis |
| Inductance of q-axis (H) | Must be a valid non-negative value | 0.0045F | Inductance of q-axis |
| Permanent magnetic flux (Wb) | Must be a valid non-negative value | 0.02159F | Permanent magnetic flux |
| Motor Parameter > Rotor inertia (kgm^2) | Must be a valid non-negative value | 0.0000028F | Rotor inertia |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at timer interrupt. |
| Input data | Name must be a valid C symbol | NULL | Structure for Speed control Input. If you set this content, Speed Control function read these data automatically. (No need to use Set API.) |
| Output data | Name must be a valid C symbol | NULL | Structure for Speed control Output. If you set this content, Speed Control function write need data automatically. (No need to use Get API.) |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Limitations

- Set the period of speed control with none-negative value.
- Set the limit of speed change step with none-negative value.
- Set the maximum speed with none-negative value.

# Examples

### Basic Example

This is a basic example of minimal use of the motor speed in an application.

```
void motor_speed_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SPEED_Open(g_motor_speed0.p_ctrl, g_motor_speed0.p_cfg);
    handle_error(err);
 /* Set speed reference before get current reference */
    (void) RM_MOTOR_SPEED_SpeedReferenceSet(g_motor_speed0.p_ctrl, 104.72F);
 /* Set position reference before get current reference
  * (Basically Exclusive to SpeedReferenceSet. This is only sample,) */
    (void) RM_MOTOR_SPEED_PositionReferenceSet(g_motor_speed0.p_ctrl,
&g_posref_sample);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set input parameter data before get current reference */
       (void) RM_MOTOR_SPEED_ParameterSet(g_motor_speed0.p_ctrl,
&g_test_speed_input);
 /* Activate Speed Process */
       (void) RM_MOTOR_SPEED_Run(g_motor_speed0.p_ctrl);
 /* Perform Speed Control Process */
       (void) RM_MOTOR_SPEED_SpeedControl(g_motor_speed0.p_ctrl);
```

```
 /* Get output parameters */
       (void) RM_MOTOR_SPEED_ParameterGet(g_motor_speed0.p_ctrl,
&g_test_speed_output);
 //

 /* Update parameters */
       (void) RM_MOTOR_SPEED_ParameterUpdate(g_motor_speed0.p_ctrl,
&g_motor_speed0.p_cfg);
    }
 /* Reset Speed Control */
    (void) RM_MOTOR_SPEED_Reset(g_motor_speed0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SPEED_Close(g_motor_speed0.p_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_speed_control_type_t |
| enum | motor_speed_openloop_damping_t |
| enum | motor_speed_flux_weaken_t |
| enum | motor_speed_less_switch_t |
| enum | motor_speed_observer_switch_t |
| enum | motor_speed_observer_select_t |
| enum | motor_speed_ctrl_status_t |

## Enumeration Type Documentation

### ◆ motor_speed_control_type_t

| enum motor_speed_control_type_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_CONTROL_TYPE_SENSORLESS | Sensorless type. |
| MOTOR_SPEED_CONTROL_TYPE_ENCODER | Encoder type. |
| MOTOR_SPEED_CONTROL_TYPE_HALL | Hall type. |
| MOTOR_SPEED_CONTROL_TYPE_INDUCTION | Induction type. |

### ◆ motor_speed_openloop_damping_t

| enum motor_speed_openloop_damping_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_OPENLOOP_DAMPING_DISABLE | Disable openloop damping. |
| MOTOR_SPEED_OPENLOOP_DAMPING_ENABLE | Enable openloop damping. |

### ◆ motor_speed_flux_weaken_t

| enum motor_speed_flux_weaken_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_FLUX_WEAKEN_DISABLE | Disable flux-weakening control. |
| MOTOR_SPEED_FLUX_WEAKEN_ENABLE | Enable flux-weakening control. |

### ◆ motor_speed_less_switch_t

| enum motor_speed_less_switch_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_LESS_SWITCH_DISABLE | Disable soft switching between open-loop mode and normal field oriented control mode. |
| MOTOR_SPEED_LESS_SWITCH_ENABLE | Enable soft switching between open-loop mode and normal field oriented control mode. |

### ◆ motor_speed_observer_switch_t

| enum motor_speed_observer_switch_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_OBSERVER_SWITCH_DISABLE | Disable speed observer. |
| MOTOR_SPEED_OBSERVER_SWITCH_ENABLE | Enable speed observer. |

### ◆ motor_speed_observer_select_t

| enum motor_speed_observer_select_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_OBSERVER_SELECT_NORMAL | Normal speed observer. |
| MOTOR_SPEED_OBSERVER_SELECT_DISTURBANCE | Disturbance speed observer. |

### ◆ motor_speed_ctrl_status_t

| enum motor_speed_ctrl_status_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_CTRL_STATUS_INIT | Speed control status is INIT. |
| MOTOR_SPEED_CTRL_STATUS_BOOT | Speed control status is BOOT. |
| MOTOR_SPEED_CTRL_STATUS_RUN | Speed control status is RUN. |

## Function Documentation

### ◆ RM_MOTOR_SPEED_Open()

fsp_err_t RM_MOTOR_SPEED_Open ( motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg* )

Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor Speed Module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

### ◆ RM_MOTOR_SPEED_Close()

fsp_err_t RM_MOTOR_SPEED_Close ( motor_speed_ctrl_t *const *p_ctrl*)

Disables specified Motor Speed Module. Implements motor_speed_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SPEED_Reset()

fsp_err_t RM_MOTOR_SPEED_Reset ( motor_speed_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SPEED_Run()

fsp_err_t RM_MOTOR_SPEED_Run ( motor_speed_ctrl_t *const *p_ctrl*)

Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully start. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SPEED_SpeedReferenceSet()

fsp_err_t RM_MOTOR_SPEED_SpeedReferenceSet ( motor_speed_ctrl_t *const *p_ctrl*, float const *speed_reference_rpm* )

Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SPEED_PositionReferenceSet()

fsp_err_t RM_MOTOR_SPEED_PositionReferenceSet ( motor_speed_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position_data* )

Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input structure pointer is NULL. |

### ◆ RM_MOTOR_SPEED_ParameterSet()

fsp_err_t RM_MOTOR_SPEED_ParameterSet ( motor_speed_ctrl_t *const *p_ctrl*, motor_speed_input_t const *const *p_st_input* )

Set Input parameters. Implements motor_speed_api_t::parameterSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data is set. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_SPEED_SpeedControl()

fsp_err_t RM_MOTOR_SPEED_SpeedControl ( motor_speed_ctrl_t *const *p_ctrl*)

Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful data calculation. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_SPEED_ParameterGet()

fsp_err_t RM_MOTOR_SPEED_ParameterGet ( motor_speed_ctrl_t *const *p_ctrl*, motor_speed_output_t *const *p_st_output* )

Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully the flag is gotten. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Output pointer is NULL. |

### ◆ RM_MOTOR_SPEED_ParameterUpdate()

| fsp_err_t RM_MOTOR_SPEED_ParameterUpdate ( motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg* ) |
|---|
| Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate.<br><br>**Return values**<br>FSP_SUCCESS — Successfully data was updated.<br>FSP_ERR_ASSERTION — Null pointer.<br>FSP_ERR_NOT_OPEN — Module is not open.<br>FSP_ERR_INVALID_ARGUMENT — Configuration parameter error. |

## 5.2.11.16 Motor Vector Control with hall sensors (rm_motor_hall)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_HALL_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MOTOR_HALL_Close (motor_ctrl_t *const p_ctrl)<br>Disables specified Motor Hall Control block. Implements motor_api_t::close. More... |
| fsp_err_t | RM_MOTOR_HALL_Reset (motor_ctrl_t *const p_ctrl)<br>Reset Motor Hall Control block. Implements motor_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_HALL_Run (motor_ctrl_t *const p_ctrl)<br>Run Motor (Start motor rotation). Implements motor_api_t::run. More... |
| fsp_err_t | RM_MOTOR_HALL_Stop (motor_ctrl_t *const p_ctrl)<br>Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_HALL_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t |

| | |
|---|---|
| | const error)<br><br>Set error information. Implements motor_api_t::errorSet. More... |
| fsp_err_t | RM_MOTOR_HALL_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm)<br><br>Set speed reference[rpm]. Implements motor_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_HALL_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)<br><br>Get current control status. Implements motor_api_t::statusGet. More... |
| fsp_err_t | RM_MOTOR_HALL_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)<br><br>Get current rotor angle. Implements motor_api_t::angleGet. More... |
| fsp_err_t | RM_MOTOR_HALL_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br><br>Get rotational speed. Implements motor_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_HALL_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)<br><br>Check the occurunce of Error. Implements motor_api_t::errorCheck. More... |
| fsp_err_t | RM_MOTOR_HALL_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)<br><br>Set position reference. Implements motor_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_HALL_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)<br><br>Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_HALL_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)<br><br>Select function. Implements motor_api_t::functionSelect. More... |

### Detailed Description

Usual control of a SPM motor on RA MCUs. This module implements the Motor Vector Control with hall sensors (rm_motor_hall).

# Overview

The motor vector control with hall sensors is used to control a motor rotation in an application. This module is meant to be used with Surface Permanent Magnet (SPM) motors and allows applications to start or stop motor rotation easily.

### Features

The motor vector control with hall sensors has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

### Target Hardware

The below figure shows an example of target hardware of this Motor Hall Module.

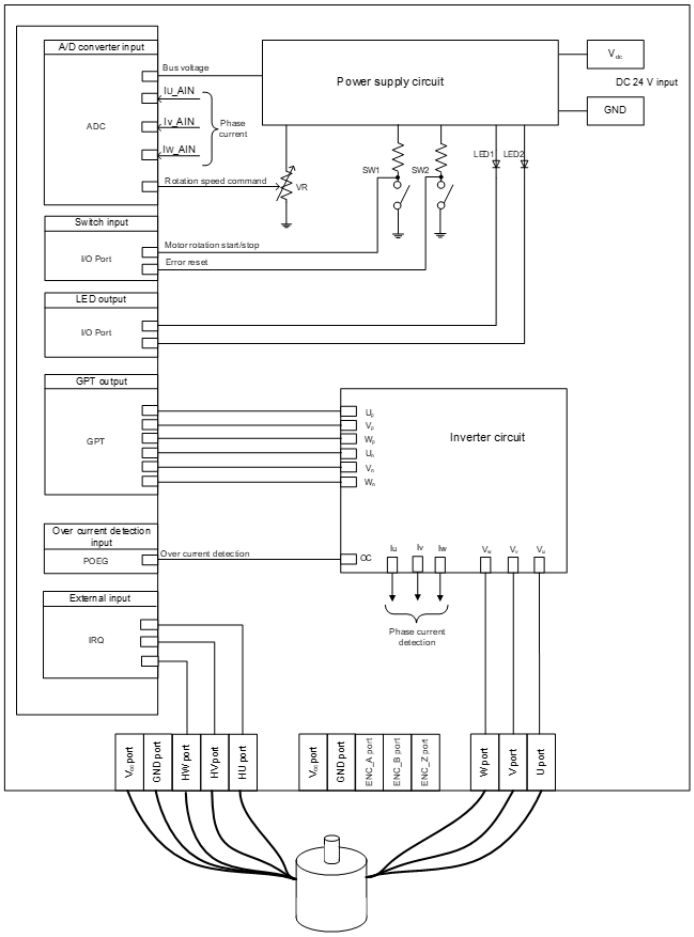



Figure 292: Example of target hardware of motor hall module

### Block Diagram

The below figure shows block diaram of vector motor control with hall sensors.

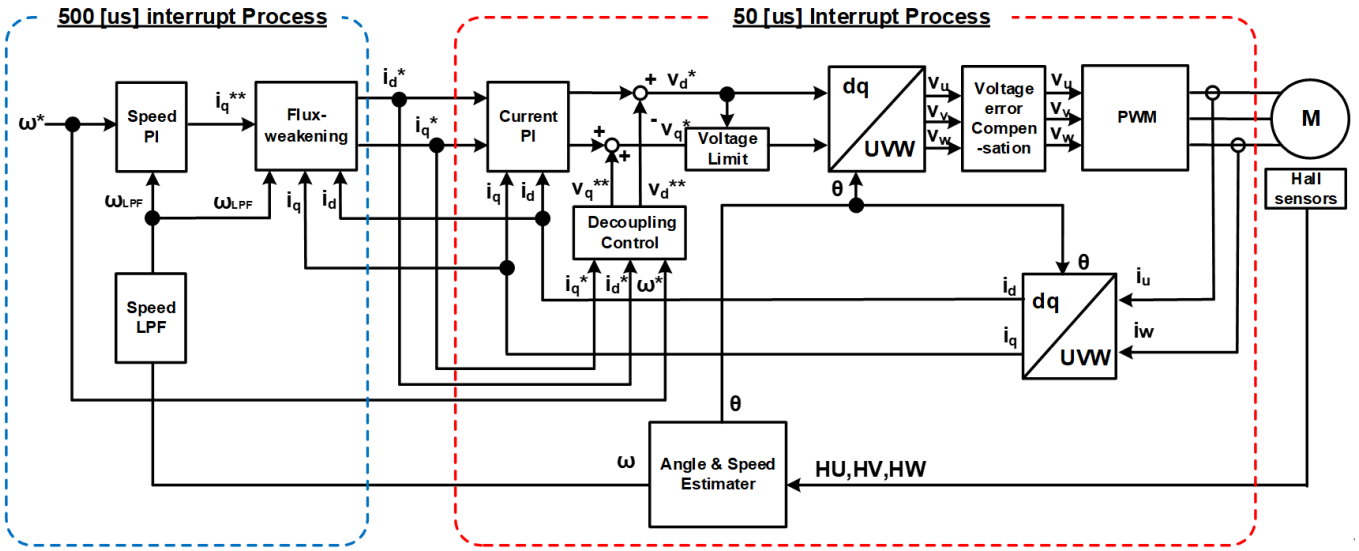



Figure 293: Block diagram of vector control with hall sensors

### Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.

$$m = \frac{V}{E}$$

m : Modulation ratio V : Reference voltage E : Inverter input voltage

Figure 294: Modulation factor

- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

$$\begin{pmatrix} V_u' \\ V_v' \\ V_w' \end{pmatrix} = \begin{pmatrix} V_u \\ V_v \\ V_w \end{pmatrix} + \Delta V \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\because \Delta V = -\frac{V_{max}+V_{min}}{2}\ ,\ V_{max} = max\{V_u, V_v, V_w\}\ ,\ V_{min} = min\{V_u, V_v, V_w\}$$

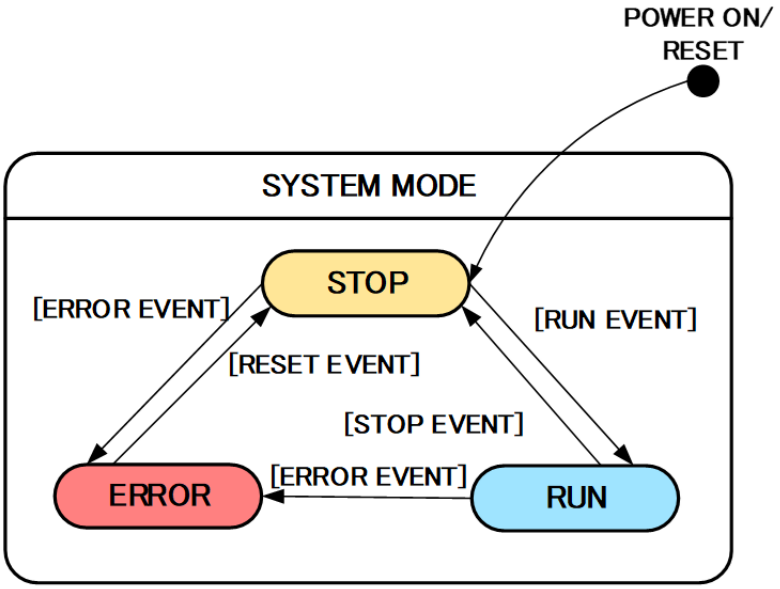
$V_u, V_v, V_w$: Command values of U-, V-, and W-phases

$V_u', V_v', V_w'$: Command values of U-, V-, and W-phases for PWM generation (modulation wave)

Figure 295: Space vector modulation

### State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".




| | | MODE | | |
|---|---|---|---|---|
| | | STOP | RUN | ERROR |
| EVENT | STOP | STOP | STOP | ERROR |
| | RUN | RUN | RUN | ERROR |
| | ERROR | ERROR | ERROR | ERROR |
| | RESET | STOP | RUN | STOP |

Figure 296: State transition diagram

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

| EVENT name | Occurrence factor |
|---|---|
| STOP | by user operation |
| RUN | by user operation |
| ERROR | when the system detects an error |
| RESET | by user operation |

**Flowchart**

The below figures show flowcharts of motor hall module.

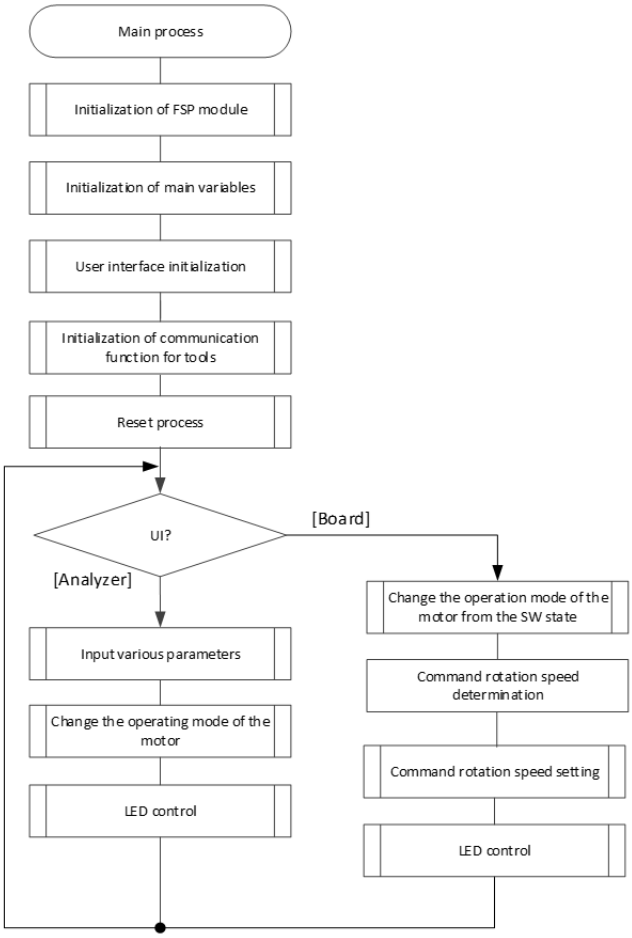



Figure 297: Main process

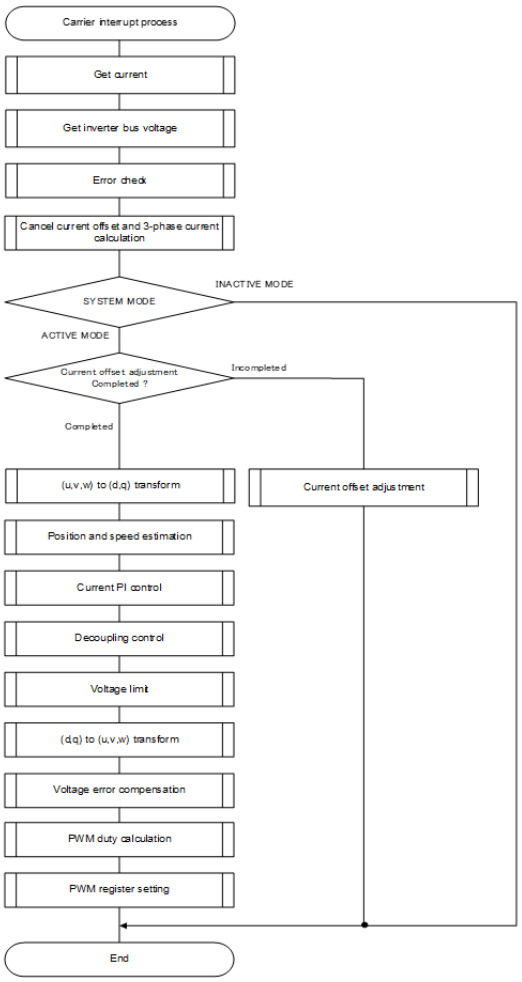



Figure 298: Current control process

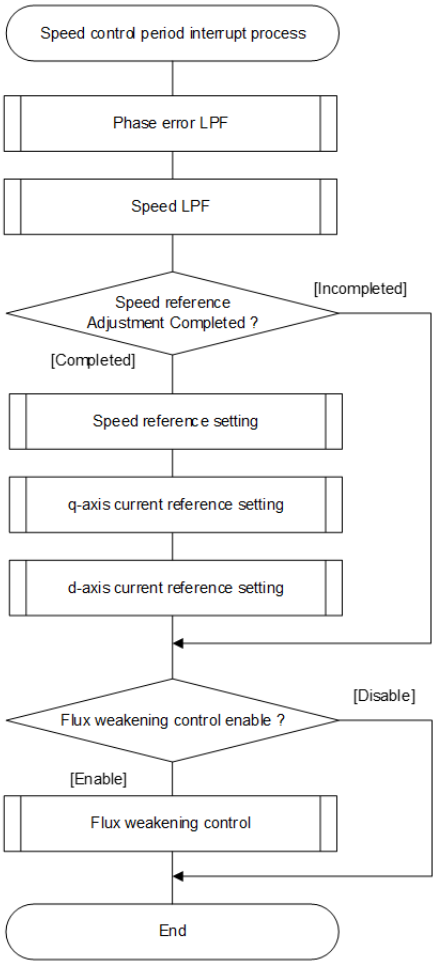



Figure 299: Speed control process

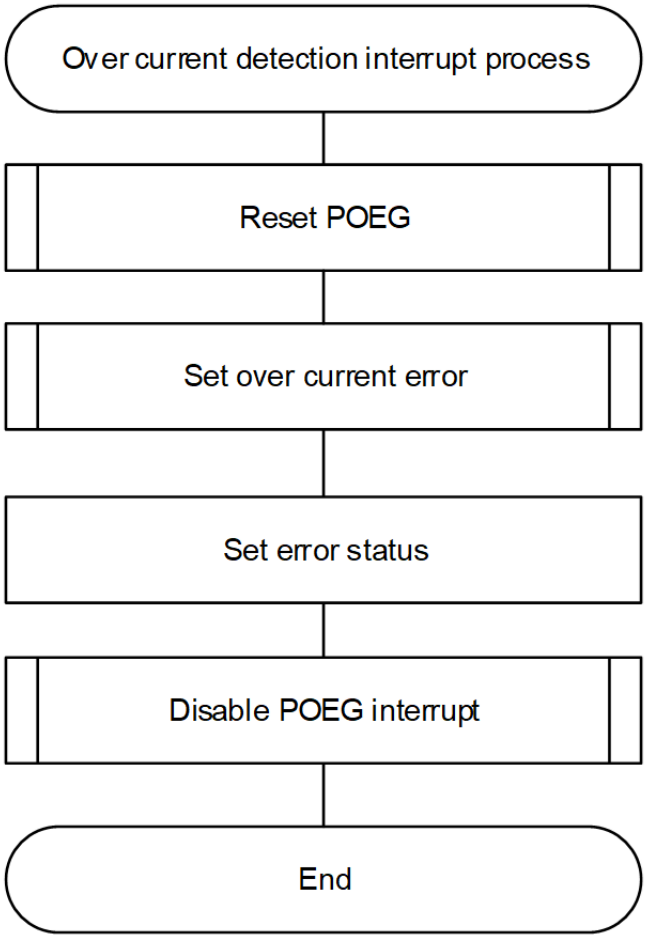



Figure 300: Over current detection interrupt process

# Configuration

### Build Time Configurations for rm_motor_hall

The following build time configurations are defined in fsp_cfg/rm_motor_hall_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor Vector Control with hall sensors(rm_motor_hall)

This module can be added to the Stacks tab via New Stack > Motor > Motor Vector Control with hall sensors(rm_motor_hall).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_motor_hall0 | Module name. |
| Limit of over current (A) | Must be a valid value | 0.42F | Limit of over current.(Detection threshold) |
| Limit of over voltage (V) | Must be a valid value | 28.0F | Limit of over voltage.(Detection threshold) |
| Limit of over speed (rpm) | Must be a valid value | 3000.0F | Limit of over speed.(Detection threshold) |
| Limit of low voltage (V) | Must be a valid value | 14.0F | Limit of low voltage.(Detection threshold) |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt. |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

### Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

## Usage Notes

### Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

## Examples

### Basic Example

This is a basic example of minimal use of the motor vector control with hall sensors in an application.

```
void motor_hall_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = RM_MOTOR_HALL_Open(g_motor_hall0_smpl.p_ctrl, g_motor_hall0_smpl.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_HALL_SpeedSet(g_motor_hall0_smpl.p_ctrl,
DEF_HALL_TEST_OVSPD_LIM);
 /* Start motor rotation */
    (void) RM_MOTOR_HALL_Run(g_motor_hall0_smpl.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_HALL_StatusGet(g_motor_hall0_smpl.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_HALL_AngleGet(g_motor_hall0_smpl.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_HALL_SpeedGet(g_motor_hall0_smpl.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_HALL_ErrorCheck(g_motor_hall0_smpl.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_HALL_Stop(g_motor_hall0_smpl.p_ctrl);
 /* When error is detected with extra Hardware */
    (void) RM_MOTOR_HALL_ErrorSet(g_motor_hall0_smpl.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset motor control (Clear error status) */
    (void) RM_MOTOR_HALL_Reset(g_motor_hall0_smpl.p_ctrl);
 /* Close motor control */
    (void) RM_MOTOR_HALL_Close(g_motor_hall0_smpl.p_ctrl);
}
```

## Function Documentation

### ◆ RM_MOTOR_HALL_Open()

fsp_err_t RM_MOTOR_HALL_Open ( motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg* )

Configure the MOTOR HALL in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_HALL_Open(g_motor_hall0_smpl.p_ctrl, g_motor_hall0_smpl.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MOTOR HALL successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

### ◆ RM_MOTOR_HALL_Close()

fsp_err_t RM_MOTOR_HALL_Close ( motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Hall Control block. Implements motor_api_t::close.

Example:

```
/* Close motor control */
   (void) RM_MOTOR_HALL_Close(g_motor_hall0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_Reset()

fsp_err_t RM_MOTOR_HALL_Reset ( motor_ctrl_t *const *p_ctrl*)

Reset Motor Hall Control block. Implements motor_api_t::reset.

Example:

```
/* Reset motor control (Clear error status) */
   (void) RM_MOTOR_HALL_Reset(g_motor_hall0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_Run()

fsp_err_t RM_MOTOR_HALL_Run ( motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_HALL_Run(g_motor_hall0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_Stop()

fsp_err_t RM_MOTOR_HALL_Stop ( motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_HALL_Stop(g_motor_hall0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_ErrorSet()

fsp_err_t RM_MOTOR_HALL_ErrorSet ( motor_ctrl_t *const *p_ctrl*, motor_error_t const *error* )

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* When error is detected with extra Hardware */
   (void) RM_MOTOR_HALL_ErrorSet(g_motor_hall0_smpl.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_SpeedSet()

fsp_err_t RM_MOTOR_HALL_SpeedSet ( motor_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_HALL_SpeedSet(g_motor_hall0_smpl.p_ctrl,
DEF_HALL_TEST_OVSPD_LIM);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_HALL_StatusGet()

fsp_err_t RM_MOTOR_HALL_StatusGet ( motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status* )

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_HALL_StatusGet(g_motor_hall0_smpl.p_ctrl, &smpl_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_HALL_AngleGet()

fsp_err_t RM_MOTOR_HALL_AngleGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad* )

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_HALL_AngleGet(g_motor_hall0_smpl.p_ctrl, &smpl_angle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_HALL_SpeedGet()

fsp_err_t RM_MOTOR_HALL_SpeedGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_HALL_SpeedGet(g_motor_hall0_smpl.p_ctrl, &smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_HALL_ErrorCheck()

fsp_err_t RM_MOTOR_HALL_ErrorCheck ( motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error* )

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_HALL_ErrorCheck(g_motor_hall0_smpl.p_ctrl, &smpl_error);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_HALL_PositionSet()

fsp_err_t RM_MOTOR_HALL_PositionSet ( motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position* )

Set position reference. Implements motor_api_t::positionSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_HALL_WaitStopFlagGet()

fsp_err_t RM_MOTOR_HALL_WaitStopFlagGet ( motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_HALL_FunctionSelect()

| fsp_err_t RM_MOTOR_HALL_FunctionSelect ( motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function* ) |
|---|
| Select function. Implements motor_api_t::functionSelect.<br><br>**Return values**<br>FSP_ERR_UNSUPPORTED — Unsupported.<br><br>***Note*** |

## 5.2.11.17 Motor return origin (rm_motor_return_origin)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_Open (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg)<br>Opens and configures the motor return origin module. Implements motor_return_origin_api_t::open. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_Close (motor_return_origin_ctrl_t *const p_ctrl)<br>Disables specified motor return origin module. Implements motor_return_origin_api_t::close. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_Start (motor_return_origin_ctrl_t *const p_ctrl)<br>Start return origin function. Implements motor_return_origin_api_t::start. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_Stop (motor_return_origin_ctrl_t *const p_ctrl)<br>Stop (Cancel) return origin function. Implements motor_return_origin_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_Reset (motor_return_origin_ctrl_t *const p_ctrl)<br>Reset variables of return origin module. Implements |

| | |
|---|---|
| | motor_return_origin_api_t::reset. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_InfoGet (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info)<br>Get information of return origin. Implements motor_return_origin_api_t::infoGet. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_DataSet (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data)<br>Set necessary data to return origin function. Implements motor_return_origin_api_t::dataSet. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_SpeedCyclic (motor_return_origin_ctrl_t *const p_ctrl)<br>Cyclic process of return origin function at speed control period. (Called at timer interrupt.) Implements motor_return_origin_api_t::speedCyclic. More... |
| fsp_err_t | RM_MOTOR_RETURN_ORIGIN_ParameterUpdate (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg)<br>Update the parameters of return origin function. Implements motor_return_origin_api_t::parameterUpdate. More... |

### Detailed Description

Search origin position process for the motor control on RA MCUs. This module implements the Motor Return Origin Function Interface.

# Overview

The motor return origin module is used to search origin position in an application. This module should be used with Renesas Motor Workbench (RMW) basically.

### Features

The Motor return origin module has below features.

- Search origin position automatically.

# Configuration

### Build Time Configurations for rm_motor_return_origin

The following build time configurations are defined in fsp_cfg/rm_motor_return_origin_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Motor > Motor return origin function (rm_motor_return_origin)**

This module can be added to the Stacks tab via New Stack > Motor > Motor return origin function (rm_motor_return_origin).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_motor_return_origin0 | Module name. |
| Using mode | Search by pushing | Search by pushing | Select using mode |
| Search speed (rpm) | Must be a valid non-negative value. | 10.0 | Speed to search origin position (rpm) |
| Acceleration of speed (rpm/sec) | Must be set 1 to 10000. | 10000.0 | Acceleration of rotation speed at return moving (rpm/sec) |
| Cyclic period of speed control (sec) | Must be a valid non-negative value. | 0.0005 | Cyclic period of speed control (sec) |
| Maximum current (A) | Must be a valid non-negative value. | 1.8 | Maximum current (A). Please set to match used motor. |
| Percentage of current to judge pushing (%) | Must be a valid non-negative value. | 30.0 | Perentage of current to judge pushing (%) |
| Pushing time (sec) | Must be a valid non-negative value. | 1.0 | Time to push the stopper (sec) |
| Degree to judge none stopper | Must be set -360 to 360. | 360.0 | When the motor runs over this value, it is judged impossible to search the stopper. |
| Degree to return | Must be a valid non-negative value. | 3.0 | Degree to return from the stopper (degree) |
| Mechanical gear ratio | Must be a valid non-negative value. | 1.0 | Mechanical gear ratio |

**Clock Configuration**

This module doesn't depend on clock setting.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

### Limitations

# Examples

### Basic Example

This is a basic example of minimal use of the Motor Return origin in an application.

```
void motor_return_origin_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_RETURN_ORIGIN_Open(&g_mtr_return_origin0_ctrl,
&g_mtr_return_origin0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start process. */
    err = RM_MOTOR_RETURN_ORIGIN_Start(&g_mtr_return_origin0_ctrl);
    temp_set_data.f_iq             = 1.0F;
    temp_set_data.f_position_degree = 180.0F;
 /* Set data to the module. */
    err = RM_MOTOR_RETURN_ORIGIN_DataSet(&g_mtr_return_origin0_ctrl, &temp_set_data);
 /* Get information from the module. */
    err = RM_MOTOR_RETURN_ORIGIN_InfoGet(&g_mtr_return_origin0_ctrl, &temp_info);
 /* Stop process. */
    err = RM_MOTOR_RETURN_ORIGIN_Stop(&g_mtr_return_origin0_ctrl);
 /* Close the module. */
    err = RM_MOTOR_RETURN_ORIGIN_Close(&g_mtr_return_origin0_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | motor_return_origin_extended_cfg_t |
| struct | motor_return_origin_pushing_t |
| struct | motor_return_origin_instance_ctrl_t |

## Data Structure Documentation

### ◆ motor_return_origin_extended_cfg_t

| struct motor_return_origin_extended_cfg_t | | |
|---|---|---|
| Extended configurations for return origin function | | |
| Data Fields | | |
| float | f_search_speed_rpm | Speed to search origin position [rpm]. |
| float | f_return_accel_rpm | Acceleration speed when return [rpm/s]. |
| float | f_speed_ctrl_period | Period of speed control [sec]. |
| float | f_maximum_current | Maximum current [A]. |
| float | f_current_limit_percent_push | Percentage of current at pushing. |
| float | f_pushing_time | Keep pushing time [sec]. |
| float | f_over_degree | Angle to judge search impossible [degree]. |
| float | f_return_degree | Return angle from pushing position [degree]. |
| float | f_mechanical_gear_ratio | Mechanical gear ratio. |

### ◆ motor_return_origin_pushing_t

| struct motor_return_origin_pushing_t | | |
|---|---|---|
| Variables for rerutn origin with pushing | | |
| Data Fields | | |
| uint32_t | u4_time_counter | Counter of speed cyclic (to judge the time) |
| float | f_sum_position | Summary of position data. |
| uint32_t | u4_sum_counter | Counter of summary. |
| float | f_move_amount | Movement amount [degree]. |
| float | f_judge_iq | q-axis current to judge pushing |
| float | f_pushing_counts | Counts to measure pushing time. |

### ◆ motor_return_origin_instance_ctrl_t

| struct motor_return_origin_instance_ctrl_t |
|---|
| Return origin function instance control block |
| Data Fields |

| | | |
|---|---|---|
| uint32_t | open | Used to determine if the module is configured. |
| motor_return_origin_start_flag_t | start_flag | start/stop flag |
| motor_return_origin_state_t | state | State number of return origin process. |
| int8_t | s1_direction | Moving direction. |
| float | f_angle_degree_on_edge | Rotor angle on the edge [degree]. |
| float | f_current_speed | Current speed. |
| float | f_origin_position_angle_degree | Searched origin position [degree]. |
| float | f_search_speed | Speed to search origin position [rad / sampling time]. |
| float | f_accel_speed | Speed acceleration. |
| float | f_position_reference_degree | Position reference [degree]. |
| motor_return_origin_pushing_t | st_pushing | Variables for pushing. |
| motor_return_origin_set_data_t | receive_data | Received data from speed(position) & current. |
| motor_return_origin_cfg_t const * | p_cfg | Pointer of configuration structure. |

## Function Documentation

### ◆ RM_MOTOR_RETURN_ORIGIN_Open()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Open ( motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_cfg_t const *const *p_cfg* )

Opens and configures the motor return origin module. Implements motor_return_origin_api_t::open .

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_RETURN_ORIGIN_Open(&g_mtr_return_origin0_ctrl,
&g_mtr_return_origin0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Motor return origin module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_RETURN_ORIGIN_Close()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Close ( motor_return_origin_ctrl_t *const *p_ctrl*)

Disables specified motor return origin module. Implements motor_return_origin_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MOTOR_RETURN_ORIGIN_Close(&g_mtr_return_origin0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_RETURN_ORIGIN_Start()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Start ( motor_return_origin_ctrl_t *const *p_ctrl*)

Start return origin function. Implements motor_return_origin_api_t::start.

Example:

```
/* Start process. */
   err = RM_MOTOR_RETURN_ORIGIN_Start(&g_mtr_return_origin0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_RETURN_ORIGIN_Stop()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Stop ( motor_return_origin_ctrl_t *const *p_ctrl*)

Stop (Cancel) return origin function. Implements motor_return_origin_api_t::stop.

Example:

```
/* Stop process. */
   err = RM_MOTOR_RETURN_ORIGIN_Stop(&g_mtr_return_origin0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully stopped (canceled). |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_RETURN_ORIGIN_Reset()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_Reset ( motor_return_origin_ctrl_t *const *p_ctrl*)

Reset variables of return origin module. Implements motor_return_origin_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully reset. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_RETURN_ORIGIN_InfoGet()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_InfoGet ( motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_info_t *const *p_info* )

Get information of return origin. Implements motor_return_origin_api_t::infoGet.

Example:

```
/* Get information from the module. */
   err = RM_MOTOR_RETURN_ORIGIN_InfoGet(&g_mtr_return_origin0_ctrl, &temp_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully get data. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_RETURN_ORIGIN_DataSet()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_DataSet ( motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_set_data_t *const *p_set_data* )

Set necessary data to return origin function. Implements motor_return_origin_api_t::dataSet.

Example:

```
/* Set data to the module. */
   err = RM_MOTOR_RETURN_ORIGIN_DataSet(&g_mtr_return_origin0_ctrl, &temp_set_data);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set data. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

### ◆ RM_MOTOR_RETURN_ORIGIN_SpeedCyclic()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_SpeedCyclic ( motor_return_origin_ctrl_t *const *p_ctrl*)

Cyclic process of return origin function at speed control period. (Called at timer interrupt.) Implements motor_return_origin_api_t::speedCyclic.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully perform the process. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MOTOR_RETURN_ORIGIN_ParameterUpdate()

fsp_err_t RM_MOTOR_RETURN_ORIGIN_ParameterUpdate ( motor_return_origin_ctrl_t *const *p_ctrl*, motor_return_origin_cfg_t const *const *p_cfg* )

Update the parameters of return origin function. Implements motor_return_origin_api_t::parameterUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data was updated. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter error. |

## 5.2.11.18 Motor vector control with induction sensor (rm_motor_induction)

Modules » Motor

### Functions

| | |
|---|---|
| fsp_err_t | RM_MOTOR_INDUCTION_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MOTOR_INDUCTION_Close (motor_ctrl_t *const p_ctrl)<br>Disables specified Motor Induction Control block. Implements motor_api_t::close. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_Reset (motor_ctrl_t *const p_ctrl)<br>Reset Motor Induction Control block. Implements motor_api_t::reset. |

| | |
|---|---|
| | More... |
| fsp_err_t | RM_MOTOR_INDUCTION_Run (motor_ctrl_t *const p_ctrl)<br>Run Motor (Start motor rotation). Implements motor_api_t::run. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_Stop (motor_ctrl_t *const p_ctrl)<br>Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error)<br>Set error information. Implements motor_api_t::errorSet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm)<br>Set speed reference[rpm]. Implements motor_api_t::speedSet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)<br>Set position reference[degree]. Implements motor_api_t::positionSet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)<br>Get current control status. Implements motor_api_t::statusGet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)<br>Get current rotor angle. Implements motor_api_t::angleGet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)<br>Get rotational speed. Implements motor_api_t::speedGet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_ErrorCheck (motor_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | uint16_t *const p_error)<br>Check the occurunce of Error. Implements motor_api_t::errorCheck. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)<br>Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_FunctionSelect (motor_ctrl_t *const p_ctrl, motor_function_select_t const function)<br>Select using function. Implements motor_api_t::functionSelect. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_InertiaEstimateStart (motor_ctrl_t *const p_ctrl)<br>Start inertia estimation function. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_InertiaEstimateStop (motor_ctrl_t *const p_ctrl)<br>Stop(Cancel) inertia estimation function. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_ReturnOriginStart (motor_ctrl_t *const p_ctrl)<br>Start return origin function. More... |
| fsp_err_t | RM_MOTOR_INDUCTION_ReturnOriginStop (motor_ctrl_t *const p_ctrl)<br>Stop(Cancel) return origin function. More... |

### Detailed Description

Control a SPM motor on RA MCUs. This module implements the Motor vector control with induction sensor (rm_motor_induction).

# Overview

The motor vector control with induction sensor is used to control motor rotation in an application. This module is meant to be used with Surface Permanent Magnet (SPM) motors and allows applications to start or stop motor rotation easily.

### Features

The motor induction module has below features.

- Start/stop motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

### Target Hardware

The below figure shows an example of target hardware of this Motor Induction Module.

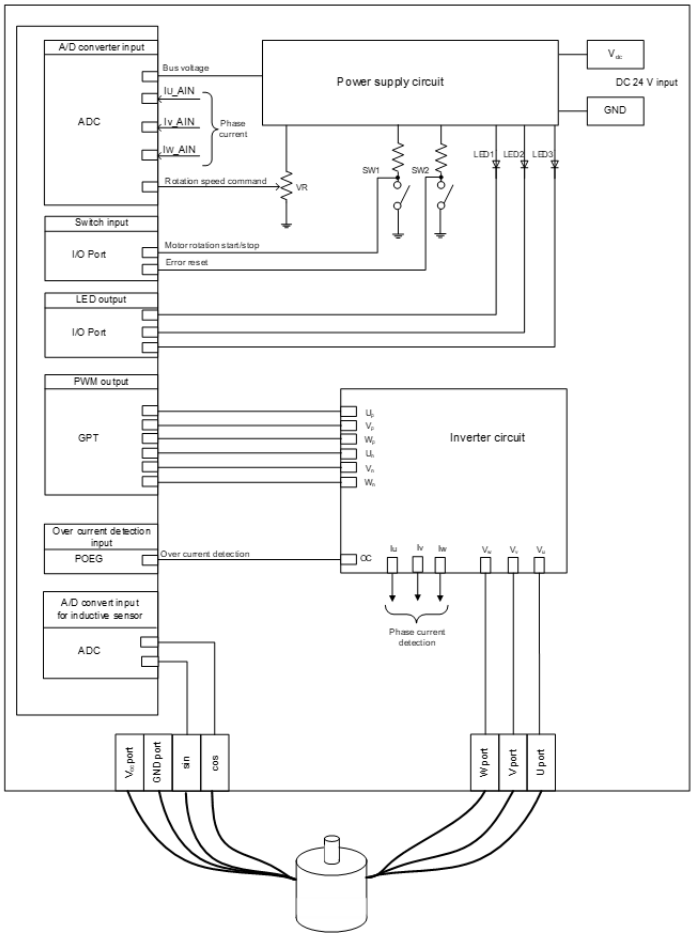



Figure 301: Example of target hardware of motor induction module

### Block Diagram

The below figures show block diaram of vector motor control with using induction sensor.

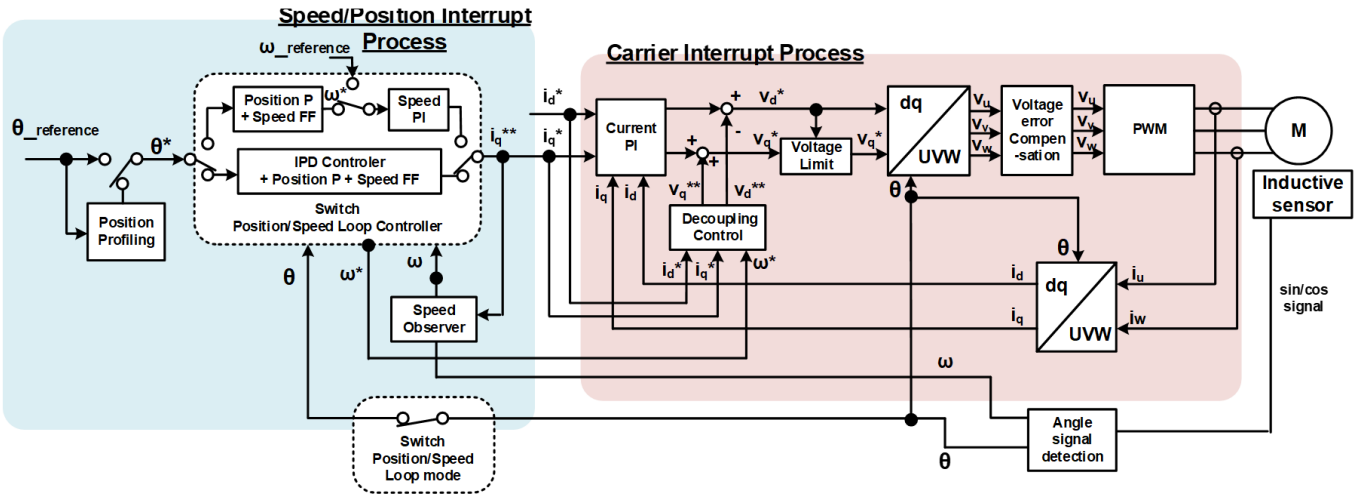



Figure 302: Block diagram of vector control with using induction sensor (PI feedback loop)

### Modulation

- Sine wave modulation The modulation factor "m" is defined as follows.

$$m = \frac{V}{E}$$

m : Modulation ratio V : Reference voltage E : Inverter input voltage

Figure 303: Modulation factor

- Space vector modulation In vector control of a permanent magnet synchronous motor, generally, the desired voltage command value of each phase is generated sinusoidally. However, if the generated value is used as-is for the modulation wave for PWM generation, voltage utilization as applied to the motor (in terms of line voltage) is limited to a maximum of 86.7 percents with respect to inverter bus voltage. As such, as shown in the following expression, the average of the maximum and minimum values is calculated for the voltage command value of each phase, and the value obtained by subtracting the average from the voltage command value of each phase is used as the modulation wave. As a result, the maximum amplitude of the modulation wave is multiplied by (square root 3)/2, while voltage utilization becomes 100 percents and line voltage is unchanged.

$$\begin{pmatrix} V_u' \\ V_v' \\ V_w' \end{pmatrix} = \begin{pmatrix} V_u \\ V_v \\ V_w \end{pmatrix} + \Delta V \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\because \Delta V = -\frac{V_{max}+V_{min}}{2},\ V_{max} = max\{V_u, V_v, V_w\},\ V_{min} = min\{V_u, V_v, V_w\}$$

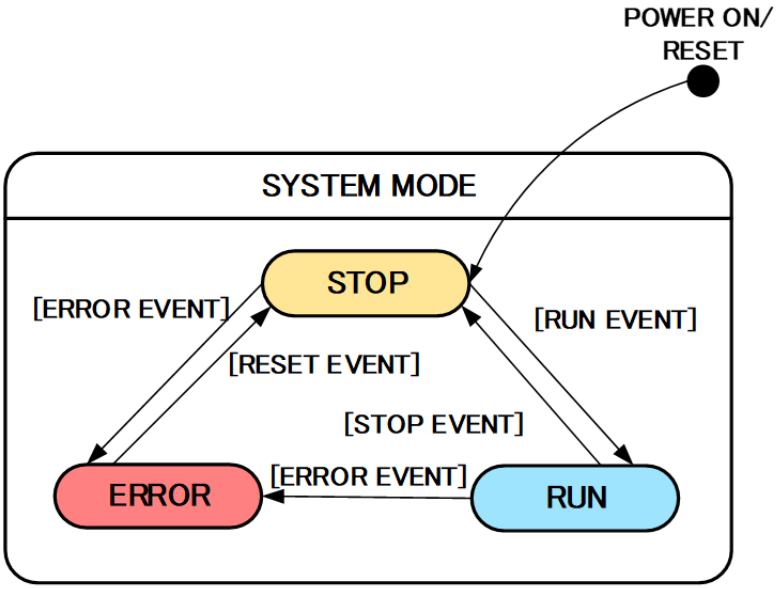
$V_u, V_v, V_w$: Command values of U-, V-, and W-phases

$V_u', V_v', V_w'$: Command values of U-, V-, and W-phases for PWM generation (modulation wave)

Figure 304: Space vector modulation

### State transition

The below figure shows a state transition diagram. Internal state is managed by "SYSTEM MODE".




| | | MODE | | |
|---|---|---|---|---|
| | | STOP | RUN | ERROR |
| EVENT | STOP | STOP | STOP | ERROR |
| | RUN | RUN | RUN | ERROR |
| | ERROR | ERROR | ERROR | ERROR |
| | RESET | STOP | RUN | STOP |

Figure 305: State transition diagram

(1) SYSTEM MODE "SYSTEM MODE" indicates the operating states of the system. The state transits on occurrence of each event (EVENT). "SYSTEM MODE" has 3 states that are motor drive stop (INACTIVE), motor drive (ACTIVE), and abnormal condition (ERROR).

(2) EVENT When "EVENT" occurs in each "SYSTEM MODE", "SYSTEM MODE" changes as shown the table in above figure, according to that "EVENT". The occurrence factors of each event are shown below.

| EVENT name | Occurrence factor |
|---|---|
| STOP | by user operation |
| RUN | by user operation |
| ERROR | when the system detects an error |
| RESET | by user operation |

**Flowchart**

The below figures show flowcharts of motor induction module.

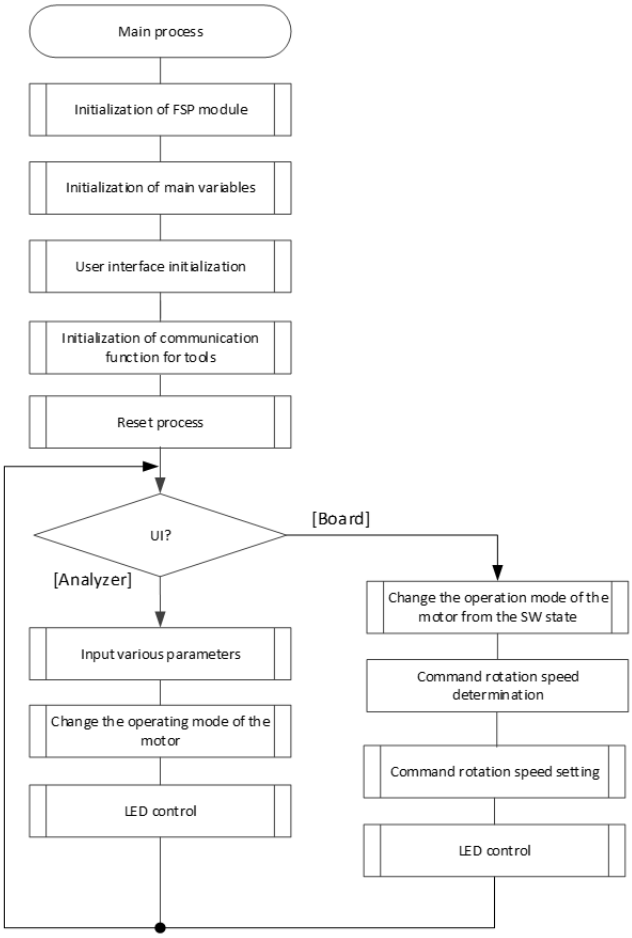



Figure 306: Main process

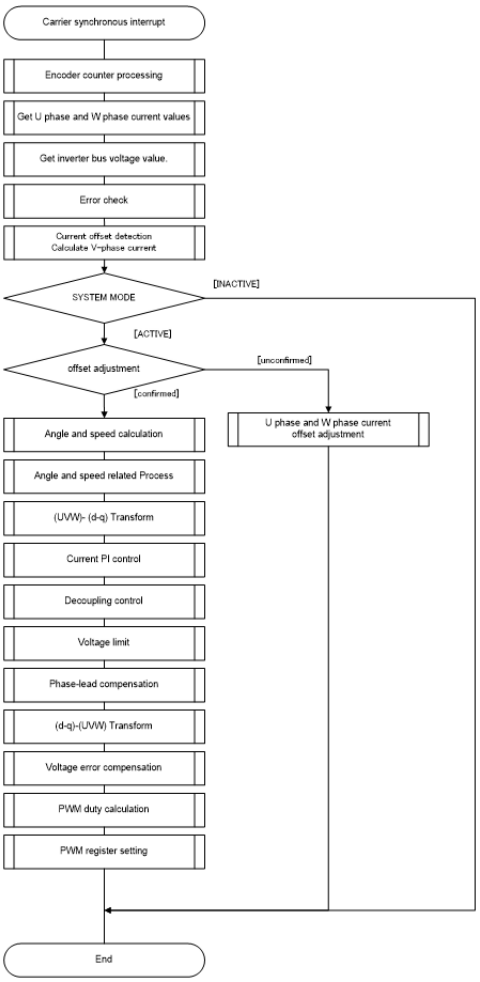



Figure 307: Current control process

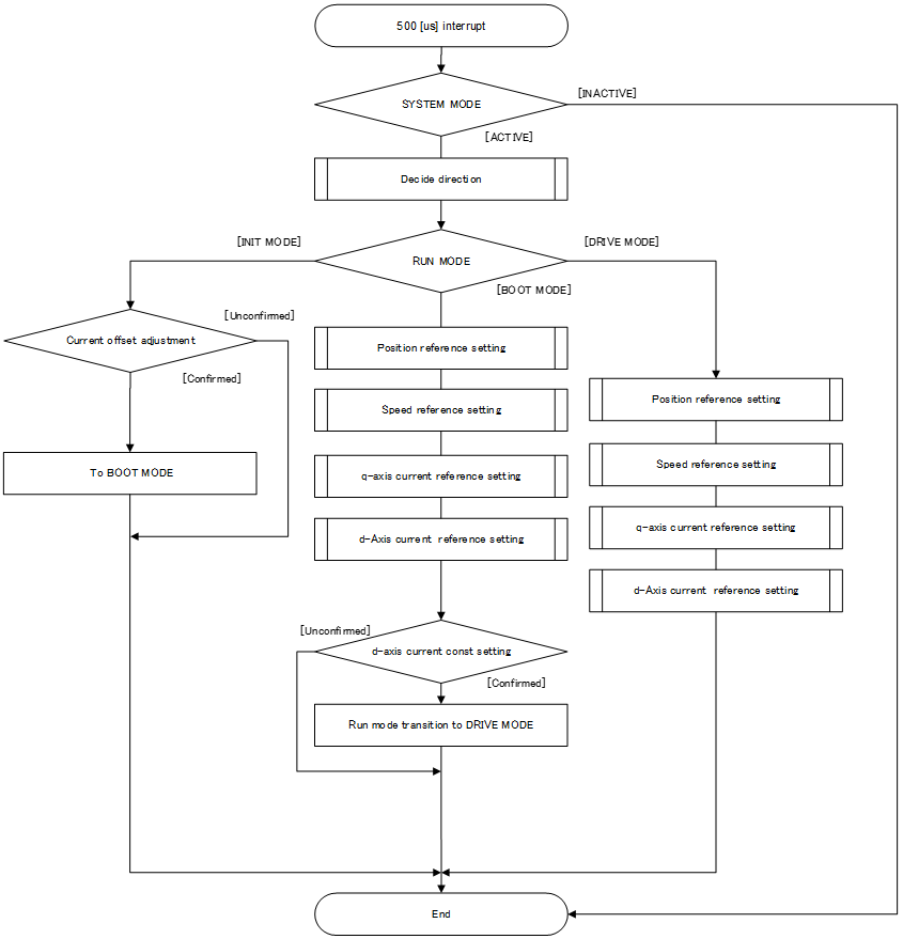



Figure 308: Speed control process

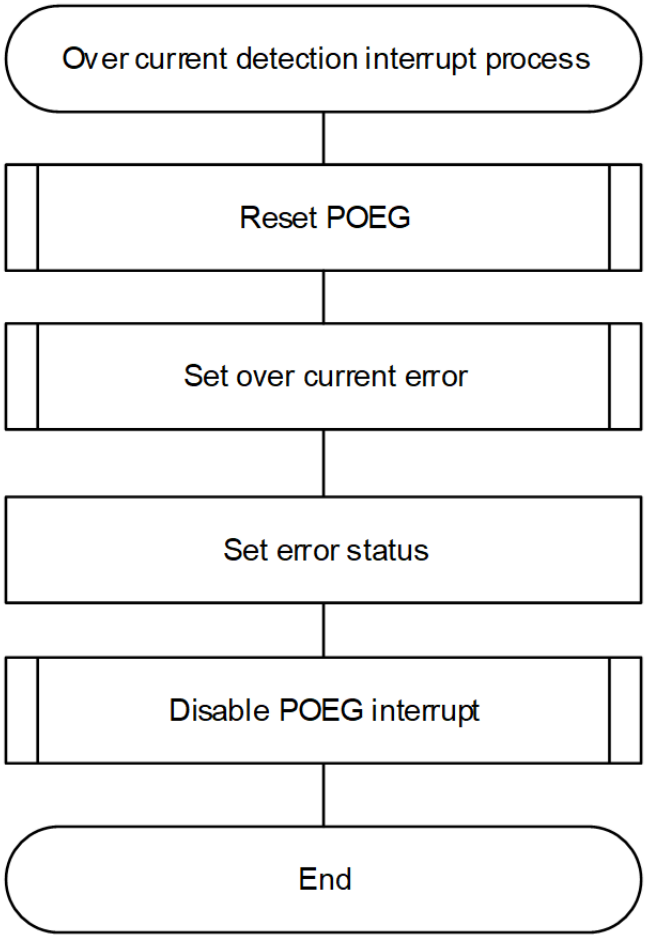



Figure 309: Over current detection interrupt process

# Configuration

### Build Time Configurations for rm_motor_induction

The following build time configurations are defined in fsp_cfg/rm_motor_induction_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Motor > Motor vector control with induction sensor (rm_motor_induction)

This module can be added to the Stacks tab via New Stack > Motor > Motor vector control with induction sensor (rm_motor_induction).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_motor_induction0 | Module name. |
| Limit of over current (A) | Must be a valid value | 2.0F | Limit of over current.(Detection threshold) |
| Limit of over voltage (V) | Must be a valid value | 28.0F | Limit of over voltage.(Detection threshold) |
| Limit of over speed (rpm) | Must be a valid value | 3000.0F | Limit of over speed.(Detection threshold) |
| Limit of low voltage (V) | Must be a valid value | 18.0F | Limit of low voltage.(Detection threshold) |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt. |

### Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

### Pin Configuration

This module does not use I/O pins. Please set used pins on configuration of each hardware modules.

## Usage Notes

### Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

## Examples

### Basic Example

This is a basic example of minimal use of the motor induction module in an application.

```
void motor_induction_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = RM_MOTOR_INDUCTION_Open(g_motor_induction0_smpl.p_ctrl,
g_motor_induction0_smpl.p_cfg);
    handle_error(err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_INDUCTION_SpeedSet(g_motor_induction0_smpl.p_ctrl,
RM_MOTOR_INDUCTION_TEST_OVER_SPEED_LIMIT);
 /* Set position reference before motor run */
    (void) RM_MOTOR_INDUCTION_PositionSet(g_motor_induction0_smpl.p_ctrl,
&g_posref_sample1);
 /* Start motor rotation */
    (void) RM_MOTOR_INDUCTION_Run(g_motor_induction0_smpl.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_INDUCTION_StatusGet(g_motor_induction0_smpl.p_ctrl,
&smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_INDUCTION_AngleGet(g_motor_induction0_smpl.p_ctrl, &smpl_angle);
 /* Get current motor rotational speed */
    (void) RM_MOTOR_INDUCTION_SpeedGet(g_motor_induction0_smpl.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_INDUCTION_ErrorCheck(g_motor_induction0_smpl.p_ctrl,
&smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_INDUCTION_Stop(g_motor_induction0_smpl.p_ctrl);
 /* If need, set extra error status */
    (void) RM_MOTOR_INDUCTION_ErrorSet(g_motor_induction0_smpl.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset motor control */
    (void) RM_MOTOR_INDUCTION_Reset(g_motor_induction0_smpl.p_ctrl);
 /* Close motor control */
    (void) RM_MOTOR_INDUCTION_Close(g_motor_induction0_smpl.p_ctrl);
}
```

## Enumerations

| | |
|---|---|
| enum | motor_induction_ctrl_t |

| | |
|---|---|
| enum | motor_induction_ctrl_event_t |

## Enumeration Type Documentation

### ◆ motor_induction_ctrl_t

| enum motor_induction_ctrl_t | |
|---|---|
| Enumerator | |
| MOTOR_INDUCTION_CTRL_STOP | Stop mode. |
| MOTOR_INDUCTION_CTRL_RUN | Run mode. |
| MOTOR_INDUCTION_CTRL_ERROR | Error mode. |

### ◆ motor_induction_ctrl_event_t

| enum motor_induction_ctrl_event_t | |
|---|---|
| Enumerator | |
| MOTOR_INDUCTION_CTRL_EVENT_STOP | Stop event. |
| MOTOR_INDUCTION_CTRL_EVENT_RUN | Run event. |
| MOTOR_INDUCTION_CTRL_EVENT_ERROR | Error event. |
| MOTOR_INDUCTION_CTRL_EVENT_RESET | Reset event. |

## Function Documentation

### ◆ RM_MOTOR_INDUCTION_Open()

fsp_err_t RM_MOTOR_INDUCTION_Open ( motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg* )

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_INDUCTION_Open(g_motor_induction0_smpl.p_ctrl,
g_motor_induction0_smpl.p_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MOTOR successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_ARGUMENT | Configuration parameter error. |

***Note***

### ◆ RM_MOTOR_INDUCTION_Close()

fsp_err_t RM_MOTOR_INDUCTION_Close ( motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Induction Control block. Implements motor_api_t::close.

Example:

```
/* Close motor control */
   (void) RM_MOTOR_INDUCTION_Close(g_motor_induction0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_Reset()

fsp_err_t RM_MOTOR_INDUCTION_Reset ( motor_ctrl_t *const *p_ctrl*)

Reset Motor Induction Control block. Implements motor_api_t::reset.

Example:

```
/* Reset motor control */
   (void) RM_MOTOR_INDUCTION_Reset(g_motor_induction0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_Run()

fsp_err_t RM_MOTOR_INDUCTION_Run ( motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_INDUCTION_Run(g_motor_induction0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_Stop()

fsp_err_t RM_MOTOR_INDUCTION_Stop ( motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_INDUCTION_Stop(g_motor_induction0_smpl.p_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_ErrorSet()

fsp_err_t RM_MOTOR_INDUCTION_ErrorSet ( motor_ctrl_t *const *p_ctrl*, motor_error_t const *error* )

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* If need, set extra error status */
   (void) RM_MOTOR_INDUCTION_ErrorSet(g_motor_induction0_smpl.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_SpeedSet()

fsp_err_t RM_MOTOR_INDUCTION_SpeedSet ( motor_ctrl_t *const *p_ctrl*, float const *speed_rpm* )

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_INDUCTION_SpeedSet(g_motor_induction0_smpl.p_ctrl,
RM_MOTOR_INDUCTION_TEST_OVER_SPEED_LIMIT);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

### ◆ RM_MOTOR_INDUCTION_PositionSet()

fsp_err_t RM_MOTOR_INDUCTION_PositionSet ( motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position* )

Set position reference[degree]. Implements motor_api_t::positionSet.

Example:

```
/* Set position reference before motor run */
   (void) RM_MOTOR_INDUCTION_PositionSet(g_motor_induction0_smpl.p_ctrl,
&g_posref_sample1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data set pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_INDUCTION_StatusGet()

fsp_err_t RM_MOTOR_INDUCTION_StatusGet ( motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status* )

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_INDUCTION_StatusGet(g_motor_induction0_smpl.p_ctrl,
&smpl_status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_INDUCTION_AngleGet()

fsp_err_t RM_MOTOR_INDUCTION_AngleGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad* )

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_INDUCTION_AngleGet(g_motor_induction0_smpl.p_ctrl, &smpl_angle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_INDUCTION_SpeedGet()

fsp_err_t RM_MOTOR_INDUCTION_SpeedGet ( motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm* )

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor rotational speed */
   (void) RM_MOTOR_INDUCTION_SpeedGet(g_motor_induction0_smpl.p_ctrl, &smpl_speed);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_INDUCTION_ErrorCheck()

fsp_err_t RM_MOTOR_INDUCTION_ErrorCheck ( motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error* )

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_INDUCTION_ErrorCheck(g_motor_induction0_smpl.p_ctrl,
&smpl_error);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Data received pointer is invalid.. |

***Note***

### ◆ RM_MOTOR_INDUCTION_WaitStopFlagGet()

fsp_err_t RM_MOTOR_INDUCTION_WaitStopFlagGet ( motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag* )

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

***Note***

### ◆ RM_MOTOR_INDUCTION_FunctionSelect()

fsp_err_t RM_MOTOR_INDUCTION_FunctionSelect ( motor_ctrl_t *const *p_ctrl*, motor_function_select_t const *function* )

Select using function. Implements motor_api_t::functionSelect.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_INDUCTION_InertiaEstimateStart()

fsp_err_t RM_MOTOR_INDUCTION_InertiaEstimateStart ( motor_ctrl_t *const *p_ctrl*)

Start inertia estimation function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_INDUCTION_InertiaEstimateStop()

fsp_err_t RM_MOTOR_INDUCTION_InertiaEstimateStop ( motor_ctrl_t *const *p_ctrl*)

Stop(Cancel) inertia estimation function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_INDUCTION_ReturnOriginStart()

fsp_err_t RM_MOTOR_INDUCTION_ReturnOriginStart ( motor_ctrl_t *const *p_ctrl*)

Start return origin function.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully resetted. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Mode unmatch |

***Note***

### ◆ RM_MOTOR_INDUCTION_ReturnOriginStop()

| fsp_err_t RM_MOTOR_INDUCTION_ReturnOriginStop ( motor_ctrl_t *const *p_ctrl*) |
|---|
| Stop(Cancel) return origin function.<br><br>**Return values**<br>FSP_SUCCESS — Successfully resetted.<br>FSP_ERR_ASSERTION — Null pointer.<br>FSP_ERR_NOT_OPEN — Module is not open.<br>FSP_ERR_INVALID_MODE — Mode unmatch<br><br>***Note*** |

## 5.2.12 Networking

Modules

### Detailed Description

Networking Modules.

### Modules

| | |
|---|---|
| | AWS Cellular Interface on GM (rm_cellular_gm_aws)<br>Middleware implementing the AWS Cellular API for GM cellular modems. |
| | AWS MQTT<br>This module provides the AWS MQTT integration documentation. |
| | AWS OTA PAL on MCUBoot (rm_aws_ota_pal_mcuboot)<br>AWS OTA PAL layer implementation for downloading firmware updates. |
| | AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs)<br>PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS. |

| | |
|---|---|
| | AWS coreHTTP<br>This module provides the AWS coreHTTP library. |
| | Azure Embedded Wireless Framework GM Port (rm_azure_ewf_gm) |
| | BLE Abstraction (rm_ble_abs)<br>Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface. |
| | BLE Driver (r_ble_balance)<br>Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface. |
| | BLE Driver (r_ble_compact)<br>Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface. |
| | BLE Driver (r_ble_extended)<br>Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface. |
| | BLE Mesh Network Modules<br>BLE Mesh Network Modules. |
| | Cellular Comm Interface on UART (rm_cellular_comm_uart_aws)<br>Middleware implementing the AWS Cellular Comm Interface for the FSP UART API. |
| | DA16XXX Transport Layer (rm_at_transport_da16xxx_uart)<br>Transport layer implementation for linking DA16XXX Drivers with Communications layer. |
| | EtherCAT SSC Port (rm_ethercat_ssc_port)<br>Middleware for EtherCAT Slave Stack Code (SSC) on RA microprocessors. This module implements the EtherCAT SSC Interface. |
| | |

| | |
|---|---|
| | Ethercat (r_ethercat_phy)<br><br>The Ethercat PHY module (r_ethercat_phy) provides an API for standard Ethercat PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface. |
| | Ethernet (r_ether)<br><br>Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface. |
| | Ethernet (r_ether_phy)<br><br>The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface. |
| | Ethernet (r_rmac)<br><br>Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface. |
| | Ethernet (r_rmac_phy)<br><br>The RMAC PHY module (r_rmac_phy) provides an API for standard Ethernet PHY communications applications that use the RMAC peripheral. It implements the Ethernet PHY Interface. |
| | Ethernet Generic PTP Timer (r_gptp)<br><br>Driver for the Ethernet peripheral on RA MCUs. This module implements the GPTP Interface. |
| | FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp)<br><br>Middleware for using TCP on RA MCUs. |
| | GTL BLE Abstraction (rm_ble_abs_gtl)<br><br>Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface. |
| | Layer3 Ethernet Switch (r_layer3_switch)<br><br>The LAYER3 SWITCH module (r_layer3_switch) provides an API for standard Ethernet communications applications that use the ESWM peripheral. It implements the Ethernet Switch Interface. |

| | |
|---|---|
| | NFC Reader Driver on PTX (rm_nfc_reader_ptx)<br>NFC Reader driver implementation using the PTX NFC module on RA MCUs. |
| | NetX Duo Ethernet Driver (rm_netxduo_ether) |
| | NetX Duo WiFi Driver (rm_netxduo_wifi) |
| | On Chip HTTP Client on DA16XXX (rm_http_onchip_da16xxx)<br>HTTP client implementation using the DA16XXX WiFi module on RA MCUs. |
| | On Chip MQTT Client on DA16XXX (rm_mqtt_onchip_da16xxx)<br>MQTT client implementation using the DA16XXX WiFi module on RA MCUs. |
| | PTP (r_ptp)<br>Driver for the PTP peripheral on RA MCUs. This module implements the PTP Interface. |
| | SPP BLE Abstraction (rm_ble_abs_spp)<br>Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface. |
| | Socket Communication Device (rm_comms_socket)<br>Middleware to implement a generic communications interface over LWIP. This module implements the Communications Middleware Interface. |
| | WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx)<br>Wifi and Socket implementation using the DA16XXX WiFi module on RA MCUs. |
| | WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex)<br>Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs. |
| | lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal) |

| | |
|---|---|
| | lwIP Ethernet Driver (rm_lwip_ether) |
| | lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos) |
| | lwIP HTTP Server<br>This module provides the lwIP HTTP Server integration documentation. |
| | lwIP MQTT<br>This module provides the lwIP MQTT integration documentation. |

#### 5.2.12.1 AWS Cellular Interface on GM (rm_cellular_gm_aws)

Modules » Networking

Middleware implementing the AWS Cellular API for GM cellular modems.

# Overview

See AWS documentation for how the Cellular API works: https://www.freertos.org/Documentation/api-ref/cellular/index.html

**Features**

The following GM modems are supported by this port:

- GM02S (https://sequans.com/products/monarch-2-gm02s/)

Please see the data sheets for hardware configuration for modems.

All APIs are supported by the GM port with the exception of the following RAT APIs:

- Cellular_SetRatPriority
- Cellular_GetRatPriority

The following URCs are supported by the GM port:

- CEREG (Cellular_RegisterUrcNetworkRegistrationEventCallback)
- SYSSTART (Cellular_RegisterModemEventCallback)
- SHUTDOWN (Cellular_RegisterModemEventCallback)
- SQNSRING (Cellular_SocketRegisterDataReadyCallback)

Support for other GM URCs can be possibly added by request in future releases.

**Limitations**

- This module does not handle module reset pins. The user should manually configure any reset pins in order to reset the module.
- The GM can only be used in buffer reception mode, not transparent mode. When calling Cellular_SocketConnect only CELLULAR_ACCESSMODE_BUFFER is supported.
- The AWS Cellular APIs don't currently allow the local port to be set for UDP. It can be set by modifying the localPort field in CellularSocketHandle_t after calling Cellular_CreateSocket but before calling Cellular_SocketConnect.

# Configuration

### AWS Cellular Interface

### Build Time Configurations for source

The following build time configurations are defined in aws/cellular_interface/cellular_config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Mobile Country Code Max Size | Value must be a positive integer | 3 | Mobile country code max size. |
| Mobile Network Node Max Size | Value must be a positive integer | 3 | Mobile network code max size. |
| Integrated Circuit Card Identity Max Size | Value must be a positive integer | 20 | Integrated circuit card identity max size. |
| International Mobile Subscriber Identity Max Size | Value must be a positive integer | 15 | International Mobile Subscriber Identity max size. |
| Firmware Version Max Size | Value must be a positive integer | 32 | Cellular module firmware version max size. |
| Hardware Version Max Size | Value must be a positive integer | 12 | Cellular module hardware version max size. |
| Serial Number Max Size | Value must be a positive integer | 12 | Cellular module serial number max size. |
| International Mobile Equipment Identity Max Size | Value must be a positive integer | 15 | International Mobile Equipment Identity number max size. |
| Registered Network operator Name Max Size | Value must be a positive integer | 32 | Registered network operator name max size. |
| Access Point Name Max Size | Value must be a positive integer | 64 | Access point name max size. |
| Packet Data Network Username Max Size | Value must be a positive integer | 32 | Packet data network username max size. |
| Packet Data Network Password Max Size | Value must be a positive integer | 32 | Packet data network password max size. |

| | | | |
|---|---|---|---|
| Data Network IP Address Max Size | Value must be a positive integer | 40 | Cellular data network IP address max size. |
| AT Command Max Size | Value must be a positive integer | 200 | Cellular AT command max size. |
| Max Number of Sockets | Value must be a positive integer | 6 | Cellular module number of socket max size. |
| Manufacturer ID Max Size | Value must be a positive integer | 20 | Cellular module manufacturer ID max size. |
| Model ID Max Size | Value must be a positive integer | 10 | Model ID max size. |
| EDRX List Max Size | Value must be a positive integer | 4 | EDRX list max size. |
| PDN Context ID Min Value | Value must be a positive integer | 1 | PDN context ID min value. |
| PDN Context ID Max Value | Value must be a positive integer | 16 | PDN context ID max value. |
| RAT Priority Count | Value must be a positive integer | 3 | RAT (radio access technology) priority count. |
| Socket Max Send Data Length (bytes) | Value must be in the range 1 - 1500 | 1460 | Socket max send data length. |
| Socket Max Receive Data Length (bytes) | Value must be in the range 1 - 1500 | 1500 | Socket max receive data length. |
| GetHostByName Support | • Enabled (default)<br>• Disabled | Enabled (default) | Enables/disable GetHostByName. |
| Comm Interface Send Timeout (ms) | Value must be a positive integer | 1000 | Cellular comm interface receive timeout in ms. |
| Comm Interface Receive Timeout (ms) | Value must be a positive integer | 50 | Cellular comm interface receive timeout in ms. |
| Static Allocation Context | • Enabled<br>• Disabled (default) | Disabled (default) | Use statically allocated context. |
| Comm Interface Static Allocation Context | • Enabled<br>• Disabled (default) | Disabled (default) | Use statically allocated context for comm interface. |
| Default RAT | • GSM<br>• WCDMA<br>• EDGE<br>• HSDPA<br>• HSUPA | CATM1 | Default Radio Access Technology. |

| | | | |
|---|---|---|---|
| | • HSDPAHSUPA<br>• LTE<br>• CATM1<br>• NBIOT | | |
| GSM Support | • Enabled<br>• Disabled (default) | Disabled (default) | This should be disabled when a modem doesn't support GSM or the AT+CGREG command. |
| Static Socket Context | • Enabled<br>• Disabled (default) | Disabled (default) | Use statically allocated socket. |
| AT Command Timeout (ms) | Value must be a positive integer | 5000 | Cellular common AT command timeout. |
| AT Command Raw Timeout (ms) | Value must be a positive integer | 5000 | Cellular AT command raw timeout. |
| AT Command Response prefix length | Value must be a positive integer | 5000 | Cellular AT command response prefix string length. |

**GM Cellular Interface Port**

**Configurations for Networking > AWS Cellular Interface on GM (rm_cellular_gm_aws)**

This module can be added to the Stacks tab via New Stack > Networking > AWS Cellular Interface on GM (rm_cellular_gm_aws).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Module Reset Pin (Port Number) | Refer to the RA Configuration tool for available options. | 00 | Specify the reset control port for the GM module. This property is only used in devassist. |
| Module Reset Pin (Pin Number) | Refer to the RA Configuration tool for available options. | 00 | Specify the reset control pin for the GM module. This property is only used in devassist. |

# Examples

### Basic Example

```
void rm_cellular_gm_aws_basic_example (void)
{
    CellularHandle_t       cellular_handle = NULL;
    CellularSimCardStatus_t sim_card_status;
 /* Initialize Cellular Modem. */
```

```
    CellularError_t err = Cellular_Init(&cellular_handle,
&g_cellular_comm_interface_on_uart);
    assert(CELLULAR_SUCCESS == err);
 /* Get the SIM card status */
    err = Cellular_GetSimCardStatus(cellular_handle, &sim_card_status);
    assert(CELLULAR_SUCCESS == err);
    assert(CELLULAR_SIM_CARD_INSERTED == sim_card_status.simCardState);
}
```

### Socket Example

```
#define SOCKET_EXAMPLE_BUFFER_SIZE (256)
#define SOCKET_EXAMPLE_TIMEOUT (UINT32_MAX)
#define SOCKET_EXAMPLE_IP ("255.255.255.255")
#define SOCKET_EXAMPLE_PORT (80)
uint32_t g_recv_event_count = 0;
void socket_example_data_ready_callback (CellularSocketHandle_t socketHandle, void *
pCallbackContext)
{
 FSP_PARAMETER_NOT_USED(socketHandle);
 FSP_PARAMETER_NOT_USED(pCallbackContext);
    g_recv_event_count++;
}
void rm_cellular_gm_aws_socket_example (void)
{
    CellularHandle_t       cellular_handle = NULL;
 char                  * p_data          = "hello_world";
 char                   recv_buffer[SOCKET_EXAMPLE_BUFFER_SIZE] = "\0";
    uint32_t               sent_data_length    = 0;
    uint32_t               received_data_length = 0;
    uint32_t               timeout_ms          = SOCKET_EXAMPLE_TIMEOUT;
    CellularSocketHandle_t socket_handle;
    CellularSocketAddress_t socket_address =
    {
       .ipAddress =
```

```
     {
           CELLULAR_IP_ADDRESS_V4,
           SOCKET_EXAMPLE_IP
      },
      .port      = SOCKET_EXAMPLE_PORT
   };
 /* Initialize Cellular Modem. */
   CellularError_t err = Cellular_Init(&cellular_handle,
&g_cellular_comm_interface_on_uart);
   assert(CELLULAR_SUCCESS == err);
 /* Create a TCP socket */
   err = Cellular_CreateSocket(cellular_handle,
                               1,
                               CELLULAR_SOCKET_DOMAIN_AF_INET,
                               CELLULAR_SOCKET_TYPE_STREAM,
                               CELLULAR_SOCKET_PROTOCOL_TCP,
                               &socket_handle);
   assert(CELLULAR_SUCCESS == err);
 /* Register the data ready callback */
   err = Cellular_SocketRegisterDataReadyCallback(cellular_handle,
                                                  socket_handle,
                                                  socket_example_data_ready_callback
,
                                                  NULL);
   assert(CELLULAR_SUCCESS == err);
   g_recv_event_count = 0;
 /* Connect the TCP socket */
   err = Cellular_SocketConnect(cellular_handle, socket_handle,
CELLULAR_ACCESSMODE_BUFFER, &socket_address);
   assert(CELLULAR_SUCCESS == err);
 /* Send data over the socket */
   err = Cellular_SocketSend(cellular_handle, socket_handle, (uint8_t *) p_data,
strlen(p_data), &sent_data_length);
   assert(CELLULAR_SUCCESS == err);
```

```
   assert(strlen(p_data) == sent_data_length);
/* Wait for data ready callback */
   timeout_ms = SOCKET_EXAMPLE_TIMEOUT;
while (timeout_ms > 0)
   {
if (g_recv_event_count > 0)
     {
break;
      }
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
   assert(0 != timeout_ms);
/* Receive data back */
   err = Cellular_SocketRecv(cellular_handle,
                             socket_handle,
                             (uint8_t *) recv_buffer,
                             SOCKET_EXAMPLE_BUFFER_SIZE,
                             &received_data_length);
   assert(CELLULAR_SUCCESS == err);
/* Close the socket */
   err = Cellular_SocketClose(cellular_handle, socket_handle);
   assert(CELLULAR_SUCCESS == err);
}
```

### Setting UDP local port

```
#define UDP_EXAMPLE_IP ("255.255.255.255")
#define UDP_EXAMPLE_REMOTE_PORT (80)
#define UDP_EXAMPLE_LOCAL_PORT (5000)
void rm_cellular_gm_aws_udp_local_port_set_example (void)
{
    CellularHandle_t       cellular_handle = NULL;
    CellularSocketHandle_t socket_handle   = NULL;
    CellularSocketAddress_t socket_address =
```

```
    {
       .ipAddress =
      {
            CELLULAR_IP_ADDRESS_V4,
            UDP_EXAMPLE_IP
       },
       .port      = UDP_EXAMPLE_REMOTE_PORT
    };
 /* Initialize Cellular Modem. */
    CellularError_t err = Cellular_Init(&cellular_handle,
&g_cellular_comm_interface_on_uart);
    assert(CELLULAR_SUCCESS == err);
 /* Create a UDP socket */
    err = Cellular_CreateSocket(cellular_handle,
                                1,
                                CELLULAR_SOCKET_DOMAIN_AF_INET,
                                CELLULAR_SOCKET_TYPE_DGRAM,
                                CELLULAR_SOCKET_PROTOCOL_UDP,
                                &socket_handle);
    assert(CELLULAR_SUCCESS == err);
 /* Set the local port */
    socket_handle->localPort = UDP_EXAMPLE_LOCAL_PORT;
 /* Connect the UDP socket */
    err = Cellular_SocketConnect(cellular_handle, socket_handle,
CELLULAR_ACCESSMODE_BUFFER, &socket_address);
    assert(CELLULAR_SUCCESS == err);
}
```

### 5.2.12.2 AWS MQTT

Modules » Networking

This module provides the AWS MQTT integration documentation.

# Overview

The AWS MQTT library can connect to either AWS or a third party MQTT broker such as Mosquitto. The documentation for the library can be found at the following link: coreMQTT.

### Features

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server

# Configuration

### Memory Usage

The AWS CoreMQTT library relies heavily on dynamic memory allocation for thread/task creation as well as other uses. Depending on the configuration it may be required to provide as much as 110k heap. To decrease this it is recommended to tweak the thread stack configuration values based on usage. Notable values are:

### FreeRTOS Thread

- General|Minimal Stack Size

### FreeRTOS Plus TCP

- Stack size in words (not bytes)

# Usage Notes

- A transport interface is required to use the CoreMQTT library (https://www.freertos.org/network-interface.html).
- For FSP, a transport interface over MbedTLS is provided. This transport interface can connect to RA ethernet, WiFi, and cellular modules via sockets wrappers for each module.
- TLS_FreeRTOS_Connect should be used before calling any CoreMQTT APIs. See the example for more information.
- Not all MQTT servers support maintaining a TLS session after a MQTT disconnect. TLS_FreeRTOS_Disconnect and TLS_FreeRTOS_Connect should be called to establish a new TLS session before attempting to reconnect with CoreMQTT APIs if the server does not support TLS session resumption.

### Limitations

- aws_clientcredential.h and aws_clientcredential_keys.h need to be added manually.
- MbedTLS must be initialized and key provisioning must be done before starting a secure connection.

### Examples

### Connection example using MbedTLS/PKCS11 Transport Interface

```
#define EXAMPLE_PDN_CONTEXT_NUMBER (1)

#define EXAMPLE_TIMEOUT_MS (5000)

#define EXAMPLE_DISABLE_SNI (false)
```

```
#define EXAMPLE_MQTT_HOST ("mqtt_host_server")
#define EXAMPLE_MQTT_HOST_PORT (8883)
#define EXAMPLE_MQTT_SEND_TIMEOUT (5000)
#define EXAMPLE_MQTT_RECEIVE_TIMEOUT (5000)
#define EXAMPLE_MQTT_TOPIC ("example_topic")
#define EXAMPLE_MQTT_CLIENT_IDENTIFIER ("client_id")
#define EXAMPLE_MQTT_KEEP_ALIVE_SECONDS (60)
#define EXAMPLE_MQTT_PAYLOAD ("helloworld")
struct NetworkContext
{
    TlsTransportParams_t * pParams;
};
static const char SERVER_CERTIFICATE_PEM[] =
 "-----BEGIN CERTIFICATE-----\n"
 "MIIDazCCAlOgAwIBAgIURabL79ayIywQv0y8SPnbZ1FYDRIwDQYJKoZIhvcNAQEL\n"
 "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
 "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAeFw0xOTA5MTEyMTIyMjZaFw0yMDA5\n"
 "MTAyMTIyMjZaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw\n"
 "HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggEiMA0GCSqGSIb3DQEB\n"
 "AQUAA4IBDwAwggEKAoIBAQDSA3h+5sT58FHgnovnQzsVHQ0H/3TsnEKwVzyBwTQl\n"
 "s4PbG6VXCWyyJWjdJ4XMH1oU8gAlxauFbwOO98Aquei4K3Pi/ynKNBeX4VJcLyE5\n"
 "Azq7nRIIwt4+OoZ5kV7v8JIoLY5i+Ktn3zq1t0y1ZmK6Uk/rRPonb+Kx7wQPx7jq\n"
 "ZIZGda+CgF6ZedidPcABuggqD1y3U2gLiRPoBhe9nN2hG60rRp7vhbWMF0pzTDXu\n"
 "BKF7XSTbhYz3pl6NeOCLh5E3t8x908Ui5W1zDN3iOysrcwQFtCiGTvzNtxSfli1+\n"
 "PugIt9Q2vlYmuz5qI+juxHftJSXO86M5SV7exqUOXP9RAgMBAAGjUzBRMB0GA1Ud\n"
 "DgQWBBQG8VNJEJUjpTKMjmrOY3XApNp5lDAfBgNVHSMEGDAWgBQG8VNJEJUjpTKM\n"
 "jmrOY3XApNp5lDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAt\n"
 "CabfjsYUnG8tt3/GDdhjsuG+SfeQe11S73pZi3+L616bPH5MNUv+LkgR/1AFEqt5\n"
 "WadKVTgzW5Ork1t7CfkYwrOHbyhyaaDPzERjMCfCcl8lQluBy6vE/lEb0hWq6XlO\n"
 "f6+8i+VKxWkSIXs2ZQqqYSOTTzAjHSsiiuE5WsC00ErvCvnC7uD6+3Y7W1uQRkFZ\n"
 "uSd9AN1ixPvAFi69FF/ymlJv6vII5GXOVDrIwdr50bMNuezMEx6qMNDADRH8iEaL\n"
 "JaSgfklczGiI1i7MPD4JTtsXOgKwxcBDAa0zQDVA5uBGEIOhva3m5X70N4iO7W0V\n"
 "eEhZekKeg3Fl3t/CXi8l\n"
 "-----END CERTIFICATE-----";
```

```
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "
```

```
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
/* Callback to handle MQTT events, user should add their own processing */
static void prvEventCallback(MQTTContext_t * pxMQTTContext,
                             MQTTPacketInfo_t       * pxPacketInfo,
                             MQTTDeserializedInfo_t * pxDeserializedInfo);
static void prvEventCallback (MQTTContext_t * pxMQTTContext,
                              MQTTPacketInfo_t       * pxPacketInfo,
                               MQTTDeserializedInfo_t * pxDeserializedInfo)
{
 FSP_PARAMETER_NOT_USED(pxMQTTContext);
 FSP_PARAMETER_NOT_USED(pxPacketInfo);
 FSP_PARAMETER_NOT_USED(pxDeserializedInfo);
}
/* Callback to get current time */
static uint32_t prvGetTimeMs(void);
static uint32_t prvGetTimeMs (void)
{
    TickType_t xTickCount = 0;
    uint32_t   ulTimeMs   = 0UL;
 /* Get the current tick count. */
    xTickCount = xTaskGetTickCount();
```

```
 /* Convert the ticks to milliseconds. */
    ulTimeMs = (uint32_t) xTickCount * (1000U / configTICK_RATE_HZ); //
NOLINT(readability-magic-numbers)
 return ulTimeMs;
}
NetworkContext_t networkContext;
void rm_aws_transport_interface_port_basic_example (void)
{
    ProvisioningParams_t params;
    NetworkCredentials_t credentials;
    TransportInterface_t transport;
    MQTTContext_t       mqtt_context;
    TlsTransportParams_t transport_params;
 static uint8_t       buffer[1024]; // NOLINT(readability-magic-numbers)
    MQTTFixedBuffer_t    networkBuffer;
    MQTTConnectInfo_t    connectInfo;
    MQTTPublishInfo_t    lwtInfo;
 bool                session_present;
 /* Open little FS flash and format in order to store keys */
    assert(FSP_SUCCESS == RM_LITTLEFS_FLASH_Open(g_rm_littlefs0.p_ctrl,
g_rm_littlefs0.p_cfg));
    assert(0 == lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
    assert(0 == lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
 /* Write the keys into a secure region in data flash. */
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientCertificateLength = sizeof(keyCLIENT_CERTIFICATE_PEM);
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.ulClientPrivateKeyLength = sizeof(keyCLIENT_PRIVATE_KEY_PEM);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
 /* Initialize network context */
```

```
   networkContext.pParams = &transport_params;
/* Setup network credentials */
   credentials.pAlpnProtos      = NULL;
   credentials.disableSni       = EXAMPLE_DISABLE_SNI;
   credentials.pRootCa          = (const unsigned char *) SERVER_CERTIFICATE_PEM;
   credentials.rootCaSize       = sizeof(SERVER_CERTIFICATE_PEM);
   credentials.pUserName       = NULL;
   credentials.userNameSize    = 0;
   credentials.pPassword       = NULL;
   credentials.passwordSize    = 0;
   credentials.pClientCertLabel = pkcs11configLABEL_DEVICE_CERTIFICATE_FOR_TLS;
   credentials.pPrivateKeyLabel = pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS;
/* Set transport interface */
   transport.pNetworkContext = &networkContext;
   transport.recv            = TLS_FreeRTOS_recv;
   transport.send            = TLS_FreeRTOS_send;
/* Connect to a MQTT host */
   TLS_FreeRTOS_Connect(&networkContext,
                        EXAMPLE_MQTT_HOST,
                        EXAMPLE_MQTT_HOST_PORT,
                        &credentials,
                        EXAMPLE_MQTT_RECEIVE_TIMEOUT,
                        EXAMPLE_MQTT_SEND_TIMEOUT);
/* Fill the values for network buffer. */
   networkBuffer.pBuffer = buffer;
   networkBuffer.size    = 1024;      // NOLINT
/* Initialize MQTT */
   assert(MQTTSuccess == MQTT_Init(&mqtt_context, &transport, prvGetTimeMs,
prvEventCallback, &networkBuffer));
/* Set connection info for MQTT session */
   connectInfo.cleanSession          = true;
   connectInfo.clientIdentifierLength = sizeof(EXAMPLE_MQTT_CLIENT_IDENTIFIER) - 1;
   connectInfo.pClientIdentifier      = EXAMPLE_MQTT_CLIENT_IDENTIFIER;
   connectInfo.keepAliveSeconds       = EXAMPLE_MQTT_KEEP_ALIVE_SECONDS;
```

```
    connectInfo.pUserName             = NULL;
    connectInfo.userNameLength         = 0U;
    connectInfo.pPassword             = NULL;
    connectInfo.passwordLength         = 0U;
 /* LWT Info. */
    lwtInfo.pTopicName      = EXAMPLE_MQTT_TOPIC;
    lwtInfo.topicNameLength = sizeof(EXAMPLE_MQTT_TOPIC) - 1;
    lwtInfo.pPayload       = EXAMPLE_MQTT_PAYLOAD;
    lwtInfo.payloadLength   = strlen(EXAMPLE_MQTT_PAYLOAD);
    lwtInfo.qos             = MQTTQoS0;
    lwtInfo.dup             = false;
    lwtInfo.retain          = false;
 /* Send MQTT CONNECT packet to broker. */
    assert(MQTTSuccess == MQTT_Connect(&mqtt_context, &connectInfo, &lwtInfo, 20000,
&session_present));
}
```

#### 5.2.12.3 AWS OTA PAL on MCUBoot (rm_aws_ota_pal_mcuboot)

Modules » Networking

AWS OTA PAL layer implementation for downloading firmware updates.

## Overview

This module provides the hardware port layer for the AWS IoT Over-the-air Update Library. Refer to the AWS OTA documentation: https://docs.aws.amazon.com/freertos/latest/userguide/integrate-ota-agent.html.

## Configuration

#### Build Time Configurations for rm_aws_ota_pal_mcuboot

The following build time configurations are defined in fsp_cfg/rm_aws_ota_pal_mcuboot_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP) | Default (BSP) | If selected code for |

| | | | |
|---|---|---|---|
| | • Enabled<br>• Disabled | | parameter checking is included in the build. |
| Filepath to Slot ID Callback Function | Must be a valid C symbol | NULL | Callback function for determining which Slot ID the image should be downloaded to. |
| OTA code signing signature algorithm | sig-sha256-ecdsa | sig-sha256-ecdsa | Code signing algorithm used by AWS to sign the downloaded image. |

**Configurations for Storage > AWS OTA PAL (rm_aws_ota_pal_mcuboot)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_aws_ota_pal_mcuboot0 | Module name. |

**Common AWS Configuration**

**Build Time Configurations for source**

The following build time configurations are defined in aws/ota_config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Version | | | |
| Major | Value must be a non-negative integer | 0 | Major Version number used when updating. |
| Minor | Value must be a non-negative integer | 0 | Minor Version number used when updating. |
| Build | Value must be a non-negative integer | 1 | Build Version number used when updating. |
| Custom ota_config.h | Manual Entry | | Add a path to your custom ota_config.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Log2 File Block Size | Value must be a non-negative integer | 11 | Log base 2 of the size of the file data block message (excluding the header). e.g. 11: 2^11 = 2048bytes |
| Self Test Response Wait (ms) | Value must be a non-negative integer | 16000 | Milliseconds to wait for the self test phase to succeed before we force reset. |
| File Request Wait (ms) | Value must be a non- | 10000 | Milliseconds to wait |

| | | | |
|---|---|---|---|
| | negative integer | | before requesting data blocks from the OTA service if nothing is happening. |
| Max Thingname Length | Value must be a non-negative integer | 64 | The maximum allowed length of the thing name used by the OTA agent. |
| Max Num Blocks Request | Value must be a non-negative integer | 1 | The maximum number of data blocks requested from OTA streaming service. |
| Max Num Request Momentum | Value must be a non-negative integer | 32 | The maximum number of requests allowed to send without a response before we abort. |
| Ota Update Status Frequency | Value must be a non-negative integer | 64 | How frequently the device will report its OTA progress to the cloud. |
| Max Num OTA Data Buffers | Value must be a non-negative integer | 1 | The number of data buffers reserved by the OTA agent. |
| Allow Downgrade | • Allowed<br>• Disallowed | Disallowed | Flag to enable booting into updates that have an identical or lower version than the current version. |
| Ota Firmware Update File Type ID | Value must be a non-negative integer | 0 | The file type id received in the job document. |
| Enabled Control Protocol | OTA Control Over MQTT | OTA Control Over MQTT | The protocol selected for OTA control operations. |
| Enabled Data Protcols | • OTA Data Over MQTT<br>• OTA Data Over HTTP<br>• OTA Data Over MQTT and HTTP | OTA Data Over MQTT | The protocol selected for OTA data operations. |
| OTA Primary Data Protocol | • OTA Data Over MQTT<br>• OTA Data Over HTTP | OTA Data Over MQTT | The preferred protocol selected for OTA data operations. |
| OTA Events Polling Timeout (ms) | Value must be a non-negative integer | 1000 | The polling timeout (milliseconds) to receive messages from |

event queue.

# Usage Notes

The current implementation utilizes MCUboot Port (rm_mcuboot_port) for switching images.

### Limitations

- Currently only the only supported signature method is sig-sha256-ecdsa.

### 5.2.12.4 AWS PKCS11 PAL on LittleFS (rm_aws_pkcs11_pal_littlefs)

Modules » Networking

PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.

# Overview

*Note*

***The PKCS#11 PAL LittleFS Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.***

# Configuration

There is no user configuration for this module

# Usage Notes

The current implementation utilizes LittleFS on Flash (rm_littlefs_flash) for storage.

### Limitations

- Credential access is not limited in any way.

### 5.2.12.5 AWS coreHTTP

Modules » Networking

This module provides the AWS coreHTTP library.

# Overview

The AWS coreHTTP library can be used to send HTTP and HTTPS requests. The documentation for the library can be found at the following link: coreHTTP.

### Features

- Secure and Non-secure HTTP requests
- Mutually authenticated connections

# Configuration

### Memory Usage

The AWS coreHTTP stack relies on dynamic memory allocation for thread/task creation as well as other uses. It is recommended to tweak the thread stack configuration values based on usage. Notable values are:

### FreeRTOS Thread

- General|Minimal Stack Size
- Memory Allocation|Total Heap Size

### FreeRTOS Plus TCP

- Stack size in words (not bytes)

# Usage Notes

- A transport interface is required to use the CoreHTTP library (https://www.freertos.org/network-interface.html).
- For FSP, a transport interface over MbedTLS is provided. This transport interface can connect to RA ethernet, WiFi, and cellular modules via sockets wrappers for each module.
- TLS_FreeRTOS_Connect should be used before calling any CoreHTTP APIs. See the example for more information.

### Limitations

- MbedTLS must be initialized and key provisioning must be done before starting a secure connection.

### Examples

### HTTPS GET request

```
/* Certificate copied from https://www.amazontrust.com/repository/AmazonRootCA1.pem
*/
static const char g_server_certificate[] = "-----BEGIN CERTIFICATE-----\n" \
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
```

```
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                   \
 "-----END CERTIFICATE-----";
/* Default settings to use if DHCP fails. */
const uint8_t g_default_ip_address[4] = {192, 168, 0, 100};
const uint8_t g_default_subnet_mask[4] = {255, 255, 255, 0};
const uint8_t g_default_gateway[4] = {192, 168, 0, 1};
const uint8_t g_default_dns[4] = {8, 8, 8, 8};
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
static NetworkInterface_t xInterfaces[1];
static NetworkEndPoint_t xEndPoints[1];
extern NetworkInterface_t * pxFillInterfaceDescriptor(BaseType_t xEMACIndex,
NetworkInterface_t * pxInterface);
#endif
void https_example_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
 /* In order to use the PKCS11 PAL, littlefs must be configured. */
 fsp_err_t fsp_err_status = RM_LITTLEFS_FLASH_Open(g_rm_littlefs0.p_ctrl,
```

```
g_rm_littlefs0.p_cfg);
    assert(FSP_SUCCESS == fsp_err_status);
 /* Reformat littlefs to ensure that data flash is in a known state. */
    assert(0 == lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /* Mount littlefs. */
    assert(0 == lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /*
  * Write the keys into data flash using the PKCS11 PAL so that they can be used
during TLS setup
  * Note that in an application this will only be done when provisioning a device
with a private key.
  * Once a device has been provisioned, the keys will persist in data flash.
  */
    ProvisioningParams_t params;
    params.pucClientPrivateKey       = (uint8_t *) g_client_private_key;
    params.pucClientCertificate      = (uint8_t *) g_client_certificate;
    params.ulClientPrivateKeyLength = sizeof(g_client_private_key);
    params.ulClientCertificateLength = sizeof(g_client_certificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    uint32_t err = (uint32_t) vAlternateKeyProvisioning(&params);
    assert(0 == err);
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
 /* Initialize the interface descriptor. */
    pxFillInterfaceDescriptor(0, xInterfaces);
    FreeRTOS_FillEndPoint(xInterfaces,
                          xEndPoints,
                          g_default_ip_address,
                          g_default_subnet_mask,
                          g_default_gateway,
                          g_default_dns,
                          g_ether0.p_cfg->p_mac_address);
 /* Initialise the TCP/IP stack. */
```

```
    FreeRTOS_IPInit_Multi();
#else
 /* Start up the network stack. */
    FreeRTOS_IPInit(g_default_ip_address,
                    g_default_subnet_mask,
                    g_default_gateway,
                    g_default_dns,
                    g_ether0.p_cfg->p_mac_address);
#endif
#if defined(ipconfigIPv4_BACKWARD_COMPATIBLE) && (ipconfigIPv4_BACKWARD_COMPATIBLE ==
0)
 while (pdFALSE == FreeRTOS_IsEndPointUp(xEndPoints))
#else
 while (pdFALSE == FreeRTOS_IsNetworkUp())
#endif
    {
       vTaskDelay(10);
    }
    NetworkCredentials_t xSocketsConfig = {0};
    TlsTransportStatus_t xNetworkStatus = TLS_TRANSPORT_SUCCESS;
    TlsTransportParams_t transport_params;
 /* Configure credentials for TLS authenticated session. */
    xSocketsConfig.pAlpnProtos = NULL;
    xSocketsConfig.disableSni = false;
    xSocketsConfig.pRootCa    = (const unsigned char *) g_server_certificate;
    xSocketsConfig.rootCaSize = sizeof(g_server_certificate);
    NetworkContext_t xNetworkContext = {0};
 /* Initialize network context */
    xNetworkContext.pParams = &transport_params;
 /* Attempt to create a authenticated TLS connection. */
    TLS_FreeRTOS_Connect(&xNetworkContext,
 "postman-echo.com",
                         HTTPS_EXAMPLE_TLS_PORT,
                         &xSocketsConfig,
```

```
                        HTTPS_EXAMPLE_TIMEOUT,
                        HTTPS_EXAMPLE_TIMEOUT);
    assert(TLS_TRANSPORT_SUCCESS == xNetworkStatus);
    TransportInterface_t xTransportInterface;
 /* Define the transport interface. */
    xTransportInterface.pNetworkContext = &xNetworkContext;
    xTransportInterface.send            = TLS_FreeRTOS_send;
    xTransportInterface.recv            = TLS_FreeRTOS_recv;
    HTTPRequestInfo_t    xRequestInfo    = {0};
    HTTPRequestHeaders_t xRequestHeaders = {0};
 /* Configure a GET request. */
    xRequestInfo.pHost    = "postman-echo.com";
    xRequestInfo.hostLen   = strlen(xRequestInfo.pHost);
    xRequestInfo.pMethod   = HTTP_METHOD_GET;
    xRequestInfo.methodLen = strlen(HTTP_METHOD_GET);
    xRequestInfo.pPath    = "/get?arg1=val1&arg2=val2";
    xRequestInfo.pathLen   = strlen(xRequestInfo.pPath);
    xRequestInfo.reqFlags = HTTP_REQUEST_KEEP_ALIVE_FLAG;
 /* Set the buffer used for storing request headers. */
 static uint8_t ucUserBuffer[HTTPS_EXAMPLE_USER_BUFFER_SIZE];
    xRequestHeaders.pBuffer   = ucUserBuffer;
    xRequestHeaders.bufferLen = sizeof(ucUserBuffer);
 /* Initialize the request. */
    HTTPStatus_t xHTTPStatus = HTTPClient_InitializeRequestHeaders(&xRequestHeaders,
&xRequestInfo);
    assert(HTTPSuccess == xHTTPStatus);
 /* Reuse the user buffer for storing the response headers. */
    HTTPResponse_t xResponse = {0};
    xResponse.pBuffer   = ucUserBuffer;
    xResponse.bufferLen = sizeof(ucUserBuffer);
 /* Send the request. */
    xHTTPStatus = HTTPClient_Send(&xTransportInterface, &xRequestHeaders, (uint8_t *)
NULL, 0, &xResponse, 0);
    assert(HTTPSuccess == xHTTPStatus);
```

```
    TLS_FreeRTOS_Disconnect(&xNetworkContext);

 /* The HTTPS request has completed. The result is stored in xResponse. */

}
```

### 5.2.12.6 Azure Embedded Wireless Framework GM Port (rm_azure_ewf_gm)

Modules » Networking

*Note*

***The Azure Embedded Wireless Framework is still in beta and not all APIs may be fully implemented yet.***

# Overview

This documentation is for the GMxxx ports of the Azure Embedded Wireless Framework.

- For more information on the framework see the documentation: https://azure.github.io/embedded-wireless-framework/html/index.html.
    - There are also various examples available for the RA6M4_EK: https://github.com/Azure/embedded-wireless-framework/tree/main/examples/EK-RA6M4.
- For more GM02S hardware information including pin and baud rate info please see the GM02S page: https://sequans.com/products/monarch-2-gm02s/.

#### Features

- Various APIs to control/access GMxxx modem features via AT commands.
- Raw AT command APIs
- IPv4 sockets using TCP/UDP
- The EWF framework is usable standalone with bare metal and ThreadX.
- The framework can also be used with NetX when coupled with a EWF NetX middleware layer.

#### Limitations

The following are the limitations of the Azure EWF Library:

- DTC is not supported. The lower level interface using R_UART relies on processing data one byte at a time via the interrupt callback and UART_EVENT_RX_CHAR.
- Server mode sockets should be supported but have not been fully tested due to APN limitations.
- There are potential issues when using the default (-Os) optimization or above with AC6. All files under embedded_wireless_framework should be set to compile with no optimization (-O0) when using AC6.
- nx_ip_status_check does not currently work for the EWF middleware.
- When using socket send functions (ewf_adapter_tcp_send and ewf_adapter_udp_send) the GM port is limited to sending only 1460 bytes at a time. Data sent using these functions should be broken into 1460 byte chunks or less.
    - When using NetX the packet size for the driver packet pool should be set to 1460

bytes or less. Using larger packet sizes will work but the NetX Middleware Driver will potentially break up the packets inefficiently.

### Unsupported NetX Duo Features

These features are handled directly on the GMxxx hardware or are not supported yet in the EWF library. NetX APIs for these features should not be used:

- IPv6 functionality (the EWF library doesn't yet support IPv6)
- ARP is handled directly by the cellular modem
- DHCP is handled directly by the cellular modem.

## Configuration

The GM02S port for the Embedded Wireless Framework can be added to the Stacks tab via New Stack > Networking > Azure EWF Adapter on GM02S.

### Build Time Configurations for Common

The following build time configurations are defined in fsp_cfg/azure/ewf/ewf.config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Enabled<br>• Disabled | Enabled | This enables checking of function parameters. When this is disabled, parameter checking code is not present and the footprint is reduced |
| Enable Logging | • Enabled<br>• Disabled | Disabled | This enables logging and the compilation of debug code. When disabled, logging and debug code is not present and the footprint is reduced. |
| Verbose Logging | • Enabled<br>• Disabled | Disabled | This enables verbose logging. Verbose logging will only work if a EWF Log Function is set and logging is enabled. |
| EWF Log Function | Manual Entry | printf(__VA_ARGS__) | Function the library uses for standard log messages when logging is enabled. |
| EWF Log Error Function | Manual Entry | printf(__VA_ARGS__) | Function the library uses for error log messages when logging is enabled. |

## Usage Notes

The basic setup for the library is as follows:

*Note*

***The message allocator (EWF_ALLOCATOR_THREADX_STATIC_DECLARE) should be at least 1500 bytes for GM***

1. Declare an allocator, interface, and adapter using appropriate macros (EWF_ALLOCATOR_THREADX_STATIC_DECLARE/EWF_ALLOCATOR_MEMORY_POOL_STATIC_DECLARE/EWF_ALLOCATOR_C_HEAP_STATIC_DECLARE, EWF_INTERFACE_RA_UART_STATIC_DECLARE, EWF_ADAPTER_RENESAS_GM02S_STATIC_DECLARE).
2. Start the adapter with ewf_adapter_start.
3. Setup modem service if necessary with functions from ewf_adapter_api_modem.h and other included api_modem_* files.
4. Set functionality with ewf_adapter_modem_functionality_set.
5. Wait for network registration (can use ewf_adapter_modem_network_registration_check to wait).
6. Do a service activate to set desired PDP context with ewf_adapter_modem_packet_service_activate. This should be called regardless of whether context is already activated as it sets an internal library context number for functions like ewf_adapter_get_ipv4_address.
7. Call other network functions as needed from desired headers (i.e. TCP functions are in ewf_adapter_api_tcp.h).

When using NetX Duo the following must be done:

- Before creating an IP instance make sure that the modem adapter is started and connected to a network.
- After creating an IP instance the adapter pointer has to be stored manually to the IP instance: g_ip.nx_ip_interface->nx_interface_additional_link_info = adapter_ptr;.
- After creating an IP instance the gateway address should be set to the IP of the modem in order to satisfy NetX Duo's internal routing algorithms (packets are actually routed by the modem hardware): nx_ip_gateway_address_set(&g_ip, RM_NETXDUO_TESTS_IP_ADDR);.

# Examples

### Basic TCP Socket Example

*Note*

*The user should choose the relevant EWF_ALLOCATOR and EWF_ADAPTER macros from this example to use based on the hardware and RTOS being used*

```
#define EWF_LOG_BUFFER_SIZE (256)

#define EWF_MODEM_NETWORK_WAIT_TIME_SECONDS (30)

#define EWF_HTTP_GET_SERVER "www.microsoft.com"

#define EWF_HTTP_GET "GET / HTTP/1.1\r\nHost:www.microsoft.com\r\n\r\n"

#define EWF_HTTP_PORT (80)

ewf_allocator * message_allocator_ptr = NULL;

ewf_interface * interface_ptr         = NULL;

ewf_adapter   * adapter_ptr          = NULL;
```

```
char ewf_log_buffer[EWF_LOG_BUFFER_SIZE];
void rm_azure_ewf_gm_example ()
{
 static uint8_t buffer[2048];       // NOLINT
    uint32_t       buffer_length = sizeof(buffer);
 /* Azure library has macros to declare/allocate structs and pointers.
  * Different ones should be used depending on Adapter and whether the library is
being used on bare metal or ThreadX */
 /* This is for the memory allocator using ThreadX block pools */
    EWF_ALLOCATOR_THREADX_STATIC_DECLARE(message_allocator_ptr, message_allocator,
12, 2048);
 /* This is for the memory allocator using static memory on bare metal */
    EWF_ALLOCATOR_MEMORY_POOL_STATIC_DECLARE(message_allocator_ptr,
message_allocator, 12, 2048);
 /* This is for the memory allocator using the heap on bare metal */
    EWF_ALLOCATOR_C_HEAP_STATIC_DECLARE(message_allocator_ptr, message_allocator, 12,
2048);
 /* This is for the communications interface using R_UART */
    EWF_INTERFACE_RA_UART_STATIC_DECLARE(interface_ptr, sci_uart);
 /* If using GM01Q */
    EWF_ADAPTER_RENESAS_GM01Q_STATIC_DECLARE(adapter_ptr, renesas_gm01q,
message_allocator_ptr, NULL, interface_ptr);
 /* If using GM02S */
    EWF_ADAPTER_RENESAS_GM02S_STATIC_DECLARE(adapter_ptr, renesas_gm02s,
message_allocator_ptr, NULL, interface_ptr);
 /* Start the adapter */
    assert(EWF_RESULT_OK == ewf_adapter_start(adapter_ptr));
 /* Set the ME functionality */
    assert(EWF_RESULT_OK == ewf_adapter_modem_functionality_set(adapter_ptr, "1"));
 /* Wait for the modem functionality to be up */
    assert(EWF_RESULT_OK ==
          ewf_adapter_modem_network_registration_check(adapter_ptr,
EWF_ADAPTER_MODEM_CMD_QUERY_EPS_NETWORK_REG,
```

```
EWF_MODEM_NETWORK_WAIT_TIME_SECONDS));
 /* Do a service activate to set context num correctly */
    ewf_adapter_modem_packet_service_activate(adapter_ptr, 1);
    ewf_socket_tcp socket_tcp = {0};
 /* Open TCP Socket */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_open(adapter_ptr, &socket_tcp));
 /* Connect to server */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_connect(&socket_tcp, EWF_HTTP_GET_SERVER,
EWF_HTTP_PORT));
 /* Send HTTP GET */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_send(&socket_tcp, (const uint8_t *)
EWF_HTTP_GET, strlen(EWF_HTTP_GET)));
 /* Receive HTTP GET response */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_receive(&socket_tcp, buffer,
&buffer_length, true));
 /* Shutdown socket connection to server */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_shutdown(&socket_tcp));
 /* Close/destroy socket */
    assert(EWF_RESULT_OK == ewf_adapter_tcp_close(&socket_tcp));
}
```

### NetX Duo Example

```
#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)
#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)
NX_PACKET_POOL g_packet_pool;
NX_IP          g_ip;
static uint8_t g_ip_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
```

```
void rm_azure_ewf_gm_netx_example ()
{
    uint32_t modem_ip_addr;
 /* Azure library has macros to declare/allocate structs and pointers.
  * Different ones should be used depending on Adapter and whether the library is
being used on bare metal or ThreadX */
 /* This is for the memory allocator using ThreadX block pools */
    EWF_ALLOCATOR_THREADX_STATIC_DECLARE(message_allocator_ptr, message_allocator,
12, 2048);
 /* This is for the communications interface using R_UART */
    EWF_INTERFACE_RA_UART_STATIC_DECLARE(interface_ptr, sci_uart);
 /* If using GM01Q */
    EWF_ADAPTER_RENESAS_GM01Q_STATIC_DECLARE(adapter_ptr, renesas_gm01q,
message_allocator_ptr, NULL, interface_ptr);
 /* If using GM02S */
    EWF_ADAPTER_RENESAS_GM02S_STATIC_DECLARE(adapter_ptr, renesas_gm02s,
message_allocator_ptr, NULL, interface_ptr);
 /* Start the adapter */
    assert(EWF_RESULT_OK == ewf_adapter_start(adapter_ptr));
 /* Set the ME functionality */
    assert(EWF_RESULT_OK == ewf_adapter_modem_functionality_set(adapter_ptr, "1"));
 /* Wait for the modem functionality to be up */
    assert(EWF_RESULT_OK ==
          ewf_adapter_modem_network_registration_check(adapter_ptr,
EWF_ADAPTER_MODEM_CMD_QUERY_EPS_NETWORK_REG,

EWF_MODEM_NETWORK_WAIT_TIME_SECONDS));
 /* Do a service activate to set context num correctly */
    ewf_adapter_modem_packet_service_activate(adapter_ptr, 1);
 /* Get modem IP */
    assert(EWF_RESULT_OK == ewf_adapter_get_ipv4_address(adapter_ptr,
&modem_ip_addr));
    UINT status;
 /* Create a packet pool */
```

```
   status = nx_packet_pool_create(&g_packet_pool,
"Packet Pool",
                               NETXDUO_EXAMPLE_PACKET_SIZE,
                               &g_packet_pool_memory[0],
                               NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
   assert(NX_SUCCESS == status);
/* Create an IP instance using EWF middleware */
   status = nx_ip_create(&g_ip,
"IP Instance",
                      modem_ip_addr,
                      IP_ADDRESS(255, 255, 255, 0),
                      &g_packet_pool,
                      nx_driver_ewf_adapter,
                      &g_ip_stack_memory[0],
sizeof(g_ip_stack_memory),
                      15);
   assert(NX_SUCCESS == status);
/* Save the adapter pointer in the IP instance */
   g_ip.nx_ip_interface->nx_interface_additional_link_info = adapter_ptr;
/* EWF requires gateway address to be set */
   status = nx_ip_gateway_address_set(&g_ip, modem_ip_addr);
   assert(NX_SUCCESS == status);
}
```

### 5.2.12.7 BLE Abstraction (rm_ble_abs)

Modules » Networking

#### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_ABS_Open (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_ABS_Close (ble_abs_ctrl_t *const p_ctrl)<br>Close the BLE channel. Implements ble_abs_api_t::close. More... |
| fsp_err_t | RM_BLE_ABS_Reset (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t |

| | |
|---|---|
| | init_callback) |
| fsp_err_t | RM_BLE_ABS_StartLegacyAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartExtendedAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartNonConnectableAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartPeriodicAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter) |
| fsp_err_t | RM_BLE_ABS_StartScanning (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter) |
| fsp_err_t | RM_BLE_ABS_CreateConnection (ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter) |
| fsp_err_t | RM_BLE_ABS_SetLocalPrivacy (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode) |
| fsp_err_t | RM_BLE_ABS_StartAuthentication (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle) |
| fsp_err_t | RM_BLE_ABS_DeleteBondInformation (ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter) |
| fsp_err_t | RM_BLE_ABS_ImportKeyInformation (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk) |
| fsp_err_t | RM_BLE_ABS_ExportKeyInformation (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk) |

### Detailed Description

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

# Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

### Features

The Bluetooth Low Energy (BLE) Abstraction module supports the following features:

- following GAP Role support
  - Central: The device that sends a connection request to the Peripheral device.
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
  - Observer : The device that scans for advertising.
  - Broadcaster : The device that sends advertising.
- LE 2M PHY
  - BLE communication is supported on the 2 Msym/s PHY.
- LE Coded PHY -Supports BLE communication on the Coded PHY. This enables communication over longer distances than 1M PHY and 2M PHY.
- LE Advertising Extensions
  - Up to four independent adverts can be executed simultaneously.
  - The size of Advertising Data/Scan Response Data has been expanded to a maximum of 1650 bytes.
  - Periodic Advertising is available.
- LE Channel Selection Algorithm #2
  - With the hopping channel selection algorithm added in Version 5.0, the machine that selects the channel It is possible.
- High Duty Cycle Non-Connectable Advertising
  - The ability to support non-connectable advertising with a minimum interval of up to 20 msec.
- LE Secure Connections
  - Elliptic curve Diffie-Hellman key sharing (ECDH) for pairing with passive eavesdropping support.
- Link Layer privacy
  - This feature avoids being tracked by other BLE devices by periodically changing the Bluetooth device address.
- Link Layer Extended Scanner Filter policies
  - Scan Filter support for Resolvable private addresses.
- LE Data Packet Length Extension
  - This function expands the packet size of BLE data communications. It is possible to scale up to 251 bytes.
- LE L2CAP Connection Oriented Channel Support
  - The ability to support communication using the L2CAP credit based flow control channel.
- Low Duty Cycle Directed Advertising
  - The ability to support the advertising of the Low Duty Cycle for reconnecting to a known device.
- LE Link Layer Topology
  - It supports both Master and Slave roles and can operate as Master when connected to one remote device and as Slave when connected to another remote device.
- LE Ping
  - This function checks whether the link is maintained or not by requesting the transmission of packets containing MIC after link encryption.

### BLE Library Configuration

There are three types of BLE Protocol Stacks, and the functions provided are different depending on the type of BLE Protocol Stack you select.

| BLE library feature | Extended | Balance | Compact |
|---|---|---|---|
| GAP Role | Central Peripheral Observer Broadcaster | Central Peripheral Observer Broadcaster | Peripheral Broadcaster |
| LE 2M PHY | Yes | Yes | No |
| LE Coded PHY | Yes | Yes | No |
| LE Advertising Extensions | Yes | No | No |
| LE Channel Selection Algorithm #2 | Yes | Yes | No |
| High Duty Cycle Non-Connectable Advertising | Yes | Yes | Yes |
| LE Secure Connections | Yes | Yes | Yes |
| Link Layer privacy | Yes | Yes | Yes |
| Link Layer Extended Scanner Filter policies | Yes | Yes | No |
| LE Data Packet Length Extension | Yes | Yes | Yes |
| LE L2CAP Connection Oriented Channel Support | Yes | No | No |
| Low Duty Cycle Directed Advertising | Yes | Yes | Yes |
| LE Link Layer Topology | Yes | Yes | No |
| LE Ping | Yes | Yes | Yes |
| 32-bit UUID Support in LE | Yes | Yes | Yes |

### Target Devices

The BLE Abstraction module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_abs

The following build time configurations are defined in fsp_cfg/rm_ble_abs_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enable<br>• Disable | Default (BSP) | Specify whether to include code for API parameter checking. Valid settings include. |
| Debug Public Address | Must be a valid device address | FF:FF:FF:50:90:74 | Public Address of firmware initial value. |
| Debug Random Address | Must be a valid device address | FF:FF:FF:FF:FF:FF | Random Address of firmware initial value. |
| Maximum number of connections | Value must be an integer between 1 and 7, and lower than the value defined in ble module. | 7 | Maximum number of connections. |
| Maximum connection data length | Value must be an integer between 27 and 251, and lower than the value defined in ble module. | 251 | Maximum connection data length. |
| Maximum advertising data length | Value must be an integer between 31 and 1650, and lower than the value defined in ble module. | 1650 | Maximum advertising data length. |
| Maximum advertising set number | Value must be an integer between 1 and 4, and lower than the value defined in ble module. | 4 | Maximum advertising set number. |
| Maximum periodic sync set number. | Value must be an integer between 1 and 2, and lower than the value defined in ble module. | 2 | Maximum periodic sync set number. |
| Store Security Data | • Disable<br>• Enable | Disable | Store Security Data in DataFlash. |
| Data Flash Block for Security Data | Value must be an integer between 0 and 7, and lower than the value defined in ble module. | 0 | Data Flash Block for Security Data Management. |
| Remote Device Bonding Number | Value must be an integer between 1 and 7, and lower than the value defined in ble module. | 7 | Number of remote device bonding information. |
| Connection Event Start | • Disable | Disable | Set Connection event |

| | | | |
|---|---|---|---|
| Notify | • Enable | | start notify enable/disable. |
| Connection Event Close Notify | • Disable<br>• Enable | Disable | Set Connection event close notify enable/disable. |
| Advertising Event Start Notify | • Disable<br>• Enable | Disable | Set Advertising event start notify enable/disable. |
| Advertising Event Close Notify | • Disable<br>• Enable | Disable | Set Advertising event close notify enable/disable. |
| Scanning Event Start Notify | • Disable<br>• Enable | Disable | Set Scanning event start notify enable/disable. |
| Scanning Event Close Notify | • Disable<br>• Enable | Disable | Set Scanning event close notify enable/disable. |
| Initiating Event Start Notify | • Disable<br>• Enable | Disable | Set Initiating event start notify enable/disable. |
| Initiating Event Close Notify | • Disable<br>• Enable | Disable | Set Initiating event close notify enable/disable. |
| RF Deep Sleep Start Notify | • Disable<br>• Enable | Disable | Set RF_DEEP_SLEEP start notify enable/disable. |
| RF Deep Sleep Wakeup Notify | • Disable<br>• Enable | Disable | Set RF_DEEP_SLEEP wakeup notify enable/disable. |
| Bluetooth dedicated clock | Value must be an integer between 0 and 15, and lower than the value defined in ble module. | 6 | Load capacitance adjustment. |
| DC-DC converter | • Disable<br>• Enable | Disable | Set DC-DC converter for RF part. |
| Slow Clock Source | • Use RF_LOCO<br>• Use External 32.768kHz | Use RF_LOCO | Set slow clock source for RF part. |
| MCU CLKOUT Port | • P109<br>• P205 | P109 | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, Set port of MCU CLKOUT. |
| MCU CLKOUT Frequency Output | • MCU CLKOUT frequency 32.768kHz | MCU CLKOUT frequency 32.768kHz | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set frequency |

| | | | |
|---|---|---|---|
| | • MCU CLKOUT frequency 16.384kHz | | output from CLKOUT of MCU part. |
| Sleep Clock Accuracy(SCA) | Value must be an integer between 0 and 500, and lower than the value defined in ble module. | 250 | When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set Sleep Clock Accuracy(SCA) for RF slow clock. |
| Transmission Power Maximum Value | • max +0dBm<br>• max +4dBm | max +4dBm | Set transmission power maximum value. |
| Transmission Power Default Value | • High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm)<br>• Mid 0dBm(Transmission Power Maximum Value = +0dBm) / 0dBm(Transmission Power Maximum Value = +4dBm)<br>• Low -18dBm(Transmission Power Maximum Value = +0dBm) / -20dBm(Transmission Power Maximum Value = +4dBm) | High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm) | Set default transmit power. Default transmit power is dependent on the configuration of Maximum transmission power(BLE_ABS_CFG_RF_DEF_TX_POW). |
| CLKOUT_RF Output | • No output<br>• 4MHz output<br>• 2MHz output<br>• 1MHz output | No output | Set CLKOUT_RF output setting. |
| RF_DEEP_SLEEP Transition | • Disable<br>• Enable | Enable | Set RF_DEEP_SLEEP transition. |
| MCU Main Clock Frequency | Value must be an integer between 1000 and 20000, and lower than the value defined in ble module. | 8000 | Set MCU Main Clock Frequency (kHz). Set clock source according to your board environment. HOCO: don't care. / Main Clock: 1000 to 20000 kHz / PLL Circuit: 4000 to 12500 kHz |

| | | | |
|---|---|---|---|
| Code Flash(ROM) Device Data Block | Value must be an integer between -1 and 255, and lower than the value defined in ble module. | 255 | Device specific data block on Code Flash (ROM). |
| Device Specific Data Flash Block | Value must be an integer between -1 and 7, and lower than the value defined in ble module. | -1 | Device specific data block on E2 Data Flash. |
| MTU Size Configured | Value must be an integer between 23 and 247, and lower than the value defined in ble module. | 247 | MTU Size configured by GATT MTU exchange procedure. |
| Timer Slot Maximum Number | Value must be an integer between 1 and 10, and lower than the value defined in ble module. | 10 | The maximum number of timer slot. |

**Configurations for Networking > BLE Abstraction (rm_ble_abs)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Abstraction (rm_ble_abs).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_ble_abs0 | Module name. |
| Gap callback | Name must be a valid C symbol | gap_cb | A user callback function must be provided if the BLE_ABS is configured to generate a GAP. If QE is used, set to NULL. |
| Vendor specific callback | Name must be a valid C symbol | vs_cb | A user callback function must be provided if the BLE_ABS is configured to generate a Vendor Specific. If QE is used, set to NULL. |
| GATT server callback parameter | Name must be a valid C symbol | gs_abs_gatts_cb_param | Set GATT server callback parameter. If QE is used, set to NULL. |
| GATT server callback | Must be a valid number | 2 | The number of GATT |

| | | | |
|---|---|---|---|
| number | | | Server callback functions. |
| GATT client callback parameter | Name must be a valid C symbol | gs_abs_gattc_cb_param | Set GATT client callback parameter. If QE is used, set to NULL. |
| GATT client callback number | Must be a valid number | 2 | The number of GATT Server callback functions. |
| Security | | | |
| Pairing parameters | Name must be a valid C symbol | gs_abs_pairing_param | Set pairing parameters. |
| IO capabilities of local device. | • BLE_GAP_IOCAP_DISPLAY_ONLY<br>• BLE_GAP_IOCAP_DISPLAY_YESNO<br>• BLE_GAP_IOCAP_KEYBOARD_ONLY<br>• BLE_GAP_IOCAP_NOINPUT_NOOUTPUT<br>• BLE_GAP_IOCAP_KEYBOARD_DISPLAY | BLE_GAP_IOCAP_NOINPUT_NOOUTPUT | Select IO capabilities of local device. |
| MITM protection policy. | • BLE_GAP_SEC_MITM_BEST_EFFORT<br>• BLE_GAP_SEC_MITM_STRICT | BLE_GAP_SEC_MITM_BEST_EFFORT | Select MITM protection policy. |
| Determine whether to accept only Secure Connections or not. | • BLE_GAP_SC_BEST_EFFORT<br>• BLE_GAP_SC_STRICT | BLE_GAP_SC_BEST_EFFORT | Select determine whether to accept only Secure Connections or not. |
| Type of keys to be distributed from local device. | • BLE_GAP_KEY_DIST_ENCKEY<br>• BLE_GAP_KEY_DIST_IDKEY<br>• BLE_GAP_KEY_DIST_SIGNKEY | | Select type of keys to be distributed from local device. |
| Type of keys which local device requests a remote device to distribute. | • BLE_GAP_KEY_DIST_ENCKEY<br>• BLE_GAP_KEY_DIST_IDKEY<br>• BLE_GAP_KEY_DIST_SIGNKEY | | Set type of keys which local device requests a remote device to distribute. |
| Maximum LTK size. | Valid range is 7 - 16 | 16 | Set Maximum LTK size. |

| | | | |
|---|---|---|---|
| Interrupts | | | |
| Callback provided when an ISR occurs | Name must be a valid C symbol | NULL | Callback provided when BLE ABS ISR occurs |

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

### Stack Size Configuration

*Note*

*When you use BLE on RTOS environment, make sure that thread stack size must be 0x1000 or more which call R_BLE_Execute.*

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the BLE_ABS:

# Examples

### BLE_ABS Basic Example

This is a basic example of minimal use of the BLE_ABS in an application.

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 'E', 'x', 'a', 'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 'T', 'E', 'S', 'T', '_', 'E', 'x', 'a',
'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
```

```
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
 ble_device_address_t local_identity_address;
    uint8_t             local_irk[BLE_GAP_IRK_SIZE];
    uint8_t             local_csrk[BLE_GAP_CSRK_SIZE];
    uint8_t            * p_local_irk = NULL;
    uint8_t             privacy_mode = BLE_GAP_NET_PRIV_MODE;
    uint8_t advertising_data[] =
    {
 /* Flags */
       0x02,
       0x01,
       (0x1a),
 /* Shortened Local Name */
       0x08,
       0x08,
       BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
    };
 /* Scan Response Data */
    uint8_t scan_response_data[] =
    {
 /* Complete Local Name */
       0x0D,
       0x09,
       BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
    };
 ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
    {
       .p_peer_address             =
NULL,
       .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
       .slow_advertising_period    =
```

```
0x0000,
       .p_advertising_data         =
advertising_data,
       .advertising_data_length    = sizeof
(advertising_data),
       .p_scan_response_data       =
scan_response_data,
       .scan_response_data_length = sizeof
(scan_response_data),
       .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
       .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
       .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
       .own_bluetooth_address      = {0},
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_STACK_ON event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
 /* Set local privacy. */
    err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_local_irk, privacy_mode);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP & g_ble_event_flag) && (--timeout >
0U))
```

```
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if (BLE_ABS_EVENT_FLAG_ADV_OFF & g_ble_event_flag)
      {
 /* Restart advertise, when stop advertising. */
      err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 if (FSP_SUCCESS == err)
      {
                g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
      }
 else if (FSP_ERR_INVALID_STATE == err)
      {
 /* BLE driver state is busy. */
                ;
      }
 else
      {
 /* Handle any errors. This function should be defined by the user. */
      assert(FSP_SUCCESS == err);
      }
      }
 else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
```

```
     {
 /* Stop advertising after a certain amount of time */
 R_BLE_GAP_StopAdv(g_advertising_handle);
     }
 else
     {
           ;
     }
 R_BLE_Execute();
   }
   time_out_handle_error(timeout);
 /* Export local key information. */
   err = RM_BLE_ABS_ExportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
 /* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
 /* Clean up & Close BLE driver */
   g_ble_event_flag = 0;
 /* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
}
```

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_REPT_IND (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_PAIRING_COMP (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT (0x01 << 5)
#define BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP (0x01 << 6)
#define BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL (0x0060)
#define BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW (0x0030)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL (0x0800)
```

```
#define BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW (0x0012)
#define BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD (0x0BB8)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD (0x0000)
#define BLE_ABS_EXAMPLE_CONNECTION_INTERVAL (0x0028)
#define BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT (0x0200)
#define BLE_ABS_EXAMPLE_DEVICE_ADDRESS 0x88, 0x88, 0x88, 0x88, 0x88, 0x88
#define BLE_ABS_EXAMPLE_IRK 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5,
0xA5
#define BLE_ABS_EXAMPLE_CSRK 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5,
0xA5
#define BLE_ABS_SCAN_FILTER_DATA_LENGTH (12)
/* Scan filter data (data type: Complete Local Name ) */
static uint8_t g_filter_data[] =
{
    BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME
};
/* Connection phy parameters */
ble_abs_connection_phy_parameter_t g_connection_phy_parameter =
{
    .connection_interval      = BLE_ABS_EXAMPLE_CONNECTION_INTERVAL, /* 50.0(ms) */
    .supervision_timeout      = BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT, /* 5,120(ms) */
    .connection_slave_latency = 0x0000,
};
/* Connection device address */
ble_device_address_t g_connection_device_address;
/* Connection parameters */
ble_abs_connection_parameter_t g_connection_parameter =
{
    .p_connection_phy_parameter_1M = &g_connection_phy_parameter,
    .p_device_address             = &g_connection_device_address,
    .filter_parameter             = BLE_GAP_INIT_FILT_USE_ADDR,
    .connection_timeout            = 0x05, /* 5(s) */
};
ble_abs_bond_information_parameter_t g_bond_information_parameter =
```

```
{
    .local_bond_information   = BLE_ABS_LOCAL_BOND_INFORMATION_ALL,
    .remote_bond_information = BLE_ABS_REMOTE_BOND_INFORMATION_ALL,
    .delete_non_volatile_area = BLE_ABS_DELETE_NON_VOLATILE_AREA_ENABLE,
    .p_address                = NULL,
    .abs_delete_bond_callback = delete_bond_cb,
};
void ble_abs_central_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    g_connection_handle = BLE_GAP_INVALID_CONN_HDL;
 ble_device_address_t local_identity_address =
    {
       .addr = {BLE_ABS_EXAMPLE_DEVICE_ADDRESS},
       .type = BLE_GAP_ADDR_PUBLIC
    };
    uint8_t local_irk[BLE_GAP_IRK_SIZE]   = {BLE_ABS_EXAMPLE_IRK};
    uint8_t local_csrk[BLE_GAP_CSRK_SIZE] = {BLE_ABS_EXAMPLE_CSRK};
 static ble_abs_scan_phy_parameter_t scan_phy_parameter =
    {
       .fast_scan_interval = BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL, /* 60.0(ms) */
       .fast_scan_window   = BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW,   /* 30.0(ms) */
       .slow_scan_interval = BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL, /* 1,280.0(ms) */
       .slow_scan_window   = BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW,   /* 11.25(ms) */
       .scan_type          = BLE_GAP_SCAN_ACTIVE
    };
 /* Scan parameters */
 ble_abs_scan_parameter_t scan_parameter =
    {
       .p_phy_parameter_1M       = &scan_phy_parameter,
       .fast_scan_period          = BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD, /* 30,000(ms)
*/
       .slow_scan_period          = BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD,
```

```
       .p_filter_data             = g_filter_data,
       .filter_data_length       = (uint16_t) BLE_ABS_SCAN_FILTER_DATA_LENGTH,
       .filter_ad_type            = 0x09,                            /* Data type:
Complete Local Name */
       .device_scan_filter_policy = BLE_GAP_SCAN_ALLOW_ADV_ALL,
       .filter_duplicate          = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Connection parameters */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
 /* Import local key information. */
    err = RM_BLE_ABS_ImportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start scanning. */
    err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while ((BLE_ABS_EVENT_FLAG_ADV_REPT_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if ((BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT & g_ble_event_flag) || (BLE_GAP_EVENT_SCAN_OFF
& g_ble_event_flag))
     {
           g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
           g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT;
 /* Start scanning. */
```

```
     err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
/* Handle any errors. This function should be defined by the user. */
     assert(FSP_SUCCESS == err);
     }
else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
     {
/* Stop scanning after a certain amount of time */
R_BLE_GAP_StopScan();
     }
else
     {
           ;
     }
R_BLE_Execute();
   }
   g_ble_event_flag = 0;
   time_out_handle_error(timeout);
   timeout = UINT16_MAX * UINT8_MAX * 8;
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Wait BLE_GAP_EVENT_CONN_IND event is notified. */
while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
   {
R_BLE_Execute();
   }
   time_out_handle_error(timeout);
   g_ble_event_flag = 0;
   timeout = UINT16_MAX * UINT8_MAX * 8;
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

```
 /* Wait BLE_GAP_EVENT_PAIRING_COMP event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_PAIRING_COMP & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Delete bonding information. */
    err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait delete_bond_cb application callback function is called. */
 while (!(BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP & g_ble_event_flag) && (--timeout >
0U))
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
 /* Clean up & Close BLE driver */
    g_ble_event_flag = 0;
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
void delete_bond_cb (st_ble_dev_addr_t * p_addr) {
    (void) p_addr;
    g_ble_event_flag = g_ble_event_flag | BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP;
}
```

## Data Structures

| | |
|---|---|
| struct | abs_advertising_parameter_t |
| struct | abs_scan_parameter_t |

| | |
|---|---|
| struct | ble_abs_instance_ctrl_t |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_abs_timer_cb_t) (uint32_t timer_hdl) |

### Enumerations

| | |
|---|---|
| enum | e_ble_timer_type_t |

## Data Structure Documentation

### ◆ abs_advertising_parameter_t

| struct abs_advertising_parameter_t | | |
|---|---|---|
| advertising set parameters structure | | |
| Data Fields | | |
| union abs_advertising_parameter_t | advertising_parameter | Advertising parameters. |
| uint32_t | advertising_status | Advertising status. |
| ble_device_address_t | remote_device_address | Remote device address for direct advertising. |

### ◆ abs_scan_parameter_t

| struct abs_scan_parameter_t | | |
|---|---|---|
| scan parameters structure | | |
| Data Fields | | |
| ble_abs_scan_parameter_t | scan_parameter | Scan parameters. |
| ble_abs_scan_phy_parameter_t | scan_phy_parameter_1M | 1M phy parameters for scan. |
| ble_abs_scan_phy_parameter_t | scan_phy_parameter_coded | Coded phy parameters for scan. */. |
| uint32_t | scan_status | |

### ◆ ble_abs_instance_ctrl_t

| struct ble_abs_instance_ctrl_t | | |
|---|---|---|
| BLE ABS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_ABS_Open() is called. | | |
| Data Fields | | |
| uint32_t | open | Indicates whether the open() API has been successfully called. |
| void * | p_context | Placeholder for user data. Passed to the user callback in |

| | | |
|---|---|---|
| | | ble_abs_callback_args_t. |
| ble_gap_application_callback_t | abs_gap_callback | GAP callback function. |
| ble_vendor_specific_application_callback_t | abs_vendor_specific_callback | Vendor specific callback function. |
| ble_abs_delete_bond_application_callback_t | abs_delete_bond_callback | Delete bond information callback function. |
| uint32_t | connection_timer_handle | Cancel a request for connection timer. |
| uint32_t | advertising_timer_handle | Advertising timer for legacy advertising. |
| uint32_t | scan_timer_handle | Scan interval timer. |
| abs_advertising_parameter_t | advertising_sets[BLE_MAX_NO_OF_ADV_SETS_SUPPORTED] | Advertising set information. |
| abs_scan_parameter_t | abs_scan | Scan information. |
| st_ble_dev_addr_t | loc_bd_addr | Local device address. |
| uint8_t | privacy_mode | Privacy mode. |
| uint32_t | set_privacy_status | Local privacy status. |
| ble_abs_timer_t | timer[BLE_ABS_CFG_TIMER_NUMBER_OF_SLOT] | |
| uint8_t | local_irk[BLE_GAP_IRK_SIZE] | |
| ble_abs_identity_address_info_t | identity_address_info | |
| uint32_t | current_timeout_ms | Current timeout. |
| uint32_t | elapsed_timeout_ms | Elapsed timeout. |
| ble_abs_cfg_t const * | p_cfg | Pointer to the BLE ABS configuration block. |

## Typedef Documentation

### ◆ ble_abs_timer_cb_t

| typedef void(* ble_abs_timer_cb_t) (uint32_t timer_hdl) |
|---|
| The timer callback invoked when the timer expired. |

## Enumeration Type Documentation

### ◆ e_ble_timer_type_t

| enum e_ble_timer_type_t | |
|---|---|
| The timer type. | |
| Enumerator | |
| BLE_TIMER_ONE_SHOT | One shot timer type |
| BLE_TIMER_PERIODIC | Periodic timer type |

## Function Documentation

### ◆ RM_BLE_ABS_Open()

fsp_err_t RM_BLE_ABS_Open ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_cfg_t const *const *p_cfg* )

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |
| FSP_ERR_INVALID_MODE | Invalid mode during open call |

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |

| | |
|---|---|
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |
| FSP_ERR_INVALID_MODE | Invalid mode during open call |

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_INVALID_CHANNEL | The channel number is invalid. |
| FSP_ERR_ALREADY_OPEN | Requested channel is already open in a different configuration. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

### ◆ RM_BLE_ABS_Close()

fsp_err_t RM_BLE_ABS_Close ( ble_abs_ctrl_t *const *p_ctrl*)

Close the BLE channel. Implements ble_abs_api_t::close.

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

### ◆ RM_BLE_ABS_Reset()

fsp_err_t RM_BLE_ABS_Reset ( ble_abs_ctrl_t *const *p_ctrl*, ble_event_cb_t *init_callback* )

This function is not implemented. To perform this function call R_BLE_Close followed by R_BLE_Open. Implements ble_abs_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function is not supported |

BLE is reset with this function. The process is carried out in the following order. R_BLE_Close() -> R_BLE_GAP_Terminate() -> R_BLE_Open() -> R_BLE_SetEvent(). The init_cb callback initializes the others (Host Stack, timer, etc...). Implements ble_abs_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel closed successfully. |
| FSP_ERR_ASSERTION | Null pointer presented. |
| FSP_ERR_NOT_OPEN | Control block not open. |

### ◆ RM_BLE_ABS_StartLegacyAdvertising()

fsp_err_t RM_BLE_ABS_StartLegacyAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_legacy_advertising_parameter_t const *const *p_advertising_parameter* )

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| | |

| | |
|---|---|
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
   err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

### ◆ RM_BLE_ABS_StartExtendedAdvertising()

fsp_err_t RM_BLE_ABS_StartExtendedAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_extend_advertising_parameter_t const *const *p_advertising_parameter* )

Start Extended Advertising after setting advertising parameters, advertising data. The extended advertising uses the advertising set whose advertising handle is 1. The advertising type is connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startExtendedAdvertising

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_UNSUPPORTED | Subordinate modules do not support this feature. |

Start Extended Advertising after setting advertising parameters, advertising data. The extended advertising uses the advertising set whose advertising handle is 1. The advertising type is connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startExtendedAdvertising

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Subordinate modules do not support this feature. |

### ◆ RM_BLE_ABS_StartNonConnectableAdvertising()

fsp_err_t RM_BLE_ABS_StartNonConnectableAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_non_connectable_advertising_parameter_t const *const *p_advertising_parameter* )

Start Non-Connectable Advertising after setting advertising parameters, advertising data. The non-connectable advertising uses the advertising set whose advertising handle is 2. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startNonConnectableAdvertising.

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_UNSUPPORTED | Feature not yet supported. |

Start Non-Connectable Advertising after setting advertising parameters, advertising data. The non-connectable advertising uses the advertising set whose advertising handle is 2. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startNonConnectableAdvertising.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_STATE | Host stack hasn't been initialized. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |

### ◆ RM_BLE_ABS_StartPeriodicAdvertising()

fsp_err_t RM_BLE_ABS_StartPeriodicAdvertising ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_periodic_advertising_parameter_t const *const *p_advertising_parameter* )

Start Periodic Advertising after setting advertising parameters, periodic advertising parameters, advertising data and periodic advertising data. The periodic advertising uses the advertising set whose advertising handle is 3. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startPeriodicAdvertising

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_UNSUPPORTED | Subordinate modules do not support this feature. |

Start Periodic Advertising after setting advertising parameters, periodic advertising parameters, advertising data and periodic advertising data. The periodic advertising uses the advertising set whose advertising handle is 3. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startPeriodicAdvertising

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_advertising_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The advertising parameter is out of range. |
| FSP_ERR_UNSUPPORTED | This feature is not supported in this configuration. |

### ◆ RM_BLE_ABS_StartScanning()

fsp_err_t RM_BLE_ABS_StartScanning ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_scan_parameter_t const *const *p_scan_parameter* )

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
   err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_scan_parameter is specified as NULL. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
   err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_scan_parameter is specified as NULL. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
   err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_scan_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The scan parameter is out of range. |
| FSP_ERR_IN_USE | This API is called in scanning. |
| FSP_ERR_BLE_ABS_NOT_FOUND | Usable timer slot not found. |
| FSP_ERR_UNSUPPORTED | This feature is not supported in this configuration. |

### ◆ RM_BLE_ABS_CreateConnection()

fsp_err_t RM_BLE_ABS_CreateConnection ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_connection_parameter_t const *const *p_connection_parameter* )

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */

   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_connection_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The create connection parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_connection_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The create connection parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_POINTER | p_connection_parameter is specified as NULL. |
| FSP_ERR_INVALID_ARGUMENT | The create connection parameter is out of range. |
| FSP_ERR_IN_USE | This API is called while creating a link by previous API call. |
| FSP_ERR_BLE_ABS_NOT_FOUND | Couldn't find a valid timer. |
| FSP_ERR_UNSUPPORTED | This feature is not supported in this configuration. |

### ◆ RM_BLE_ABS_SetLocalPrivacy()

fsp_err_t RM_BLE_ABS_SetLocalPrivacy ( ble_abs_ctrl_t *const *p_ctrl*, uint8_t const *const *p_lc_irk*, uint8_t *privacy_mode* )

Generate a IRK, add it to the resolving list, set privacy mode and enable RPA function. Register vendor specific callback function, if IRK is generated by this function. After configuring local device privacy, BLE_GAP_ADDR_RPA_ID_PUBLIC is specified as own device address in theadvertising/scan/create connection API. Implements ble_abs_api_t::setLocalPrivacy

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | The privacy_mode parameter is out of range. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Generate a IRK, add it to the resolving list, set privacy mode and enable RPA function. Register vendor specific callback function, if IRK is generated by this function. After configuring local device privacy, BLE_GAP_ADDR_RPA_ID_PUBLIC is specified as own device address in theadvertising/scan/create connection API. Implements ble_abs_api_t::setLocalPrivacy

Example:

```
/* Set local privacy. */
   err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_local_irk, privacy_mode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl is specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | The privacy_mode parameter is out of range. |

### ◆ RM_BLE_ABS_StartAuthentication()

fsp_err_t RM_BLE_ABS_StartAuthentication ( ble_abs_ctrl_t *const *p_ctrl*, uint16_t *connection_handle* )

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl or connection_handle are specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
```

**Return values**

| | |
|---|---|
| FSP_ERR_ASSERTION | p_instance_ctrl or connection_handle are specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_UNSUPPORTED | Function is not supported |

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | p_instance_ctrl or connection_handle are specified as NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_ARGUMENT | The connection handle parameter is out of range. |

### ◆ RM_BLE_ABS_DeleteBondInformation()

fsp_err_t RM_BLE_ABS_DeleteBondInformation ( ble_abs_ctrl_t *const *p_ctrl*, ble_abs_bond_information_parameter_t const *const *p_bond_information_parameter* )

Delete bonding information from BLE stack and storage. Implements ble_abs_api_t::deleteBondInformation.

Example:

```
/* Delete bonding information. */
   err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation was successful |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_bond_information_parameter is NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |

Delete bonding information from BLE stack and storage. Implements ble_abs_api_t::deleteBondInformation.

Example:

```
/* Delete bonding information. */
   err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation was successful |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. |

| | |
|---|---|
| FSP_ERR_INVALID_POINTER | The parameter p_bond_information_parameter is NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |

Delete bonding information from BLE stack and storage. Implements ble_abs_api_t::deleteBondInformation.

Example:

```
/* Delete bonding information. */
   err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_bond_information_parameter is NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |

### ◆ RM_BLE_ABS_ImportKeyInformation()

fsp_err_t RM_BLE_ABS_ImportKeyInformation ( ble_abs_ctrl_t *const *p_ctrl*, ble_device_address_t * *p_local_identity_address*, uint8_t * *p_local_irk*, uint8_t * *p_local_csrk* )

Import key information to BLE stack and storage. Implements ble_abs_api_t::importKeyInformation.

Example:

```
/* Import local key information. */
   err = RM_BLE_ABS_ImportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_local_identity_address, p_local_irk or p_local_csrk is NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_INVALID_HW_CONDITION | Failure to access internal storage. |
| FSP_ERR_UNSUPPORTED | Not supported in this configuration. |

### ◆ RM_BLE_ABS_ExportKeyInformation()

fsp_err_t RM_BLE_ABS_ExportKeyInformation ( ble_abs_ctrl_t *const *p_ctrl*, ble_device_address_t * *p_local_identity_address*, uint8_t * *p_local_irk*, uint8_t * *p_local_csrk* )

Export key information to BLE stack and storage. Implements ble_abs_api_t::exportKeyInformation.

Example:

```
/* Export local key information. */
   err = RM_BLE_ABS_ExportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_local_identity_address, p_local_irk or p_local_csrk is NULL. |
| FSP_ERR_NOT_OPEN | Control block not open. |
| FSP_ERR_BUFFER_EMPTY | Dynamic memory allocation failed. |
| FSP_ERR_OUT_OF_MEMORY | Failure to access internal storage. |
| FSP_ERR_NOT_INITIALIZED | Not initialized internal storage. |
| FSP_ERR_UNSUPPORTED | Not supported in this configuration. |

#### 5.2.12.8 BLE Driver (r_ble_balance)

Modules » Networking

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

# Overview

The Bluetooth Low Energy (BLE) library in Balance configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

**Features**

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.

    - Add an event in the BLE protocol stack internal queue.
- GAP
    - Initialization of the Host stack.
    - Start/Stop Advertising.
    - Start/Stop Scan.
    - Connect/Disconnect a link.
    - Initiate/Respond a pairing request.
- GATT Common
    - Get MTU size.
- GATT Server
    - Initialization of GATT Server.
    - Notification/Indication.
- GATT Client
    - Discovery services, characteristics.
    - Read/Write characteristic.
- Vendor Specific
    - DTM.
    - Set/Get transmit power.
    - Set/Get BD_ADDR.

### Supported functions

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

| BLE library feature | Extended | Balance | Compact |
|---|---|---|---|
| Common API | Supported | **Supported** | Supported |
| GAP API | Supported | ***1 Limited** | *1 *2 Limited |
| GATT Common API | Supported | **Supported** | Supported |
| GATT Client API | Supported | **Supported** | Supported |
| GATT Client API | Supported | **Supported** | Supported |
| L2CAP API | Supported | **Not Supported** | Not supported |
| Vendor Specific API | *3 Limited | ***3 Limited** | *3 Limited |

*Note*

1. *This configuration dose not support LE Advertising Extensions functionality APIs.*
2. *This configuration dose not support Central and Observer functionality APIs.*
3. *This configuration dose not support vender specific firmware update functionality APIs.*

### Target Devices

The Renesas BLE Library supports the following devices.

- RA4W1

# Configuration

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

Figure shows the software structure of the BLE FSP module.

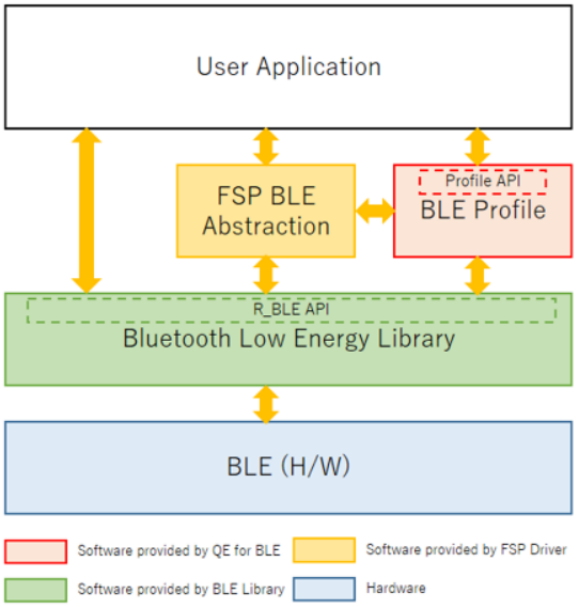



Figure 310: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

### Initialize the BLE protocol stack

*Note*

*It takes around 250msec to initialize BLE protocol stack.*
*R_BLE_Open API will be occupied MCU resources during the initialization.*

### Limitations

Developers should be aware of the following limitations when using the ble:

*Note*

*- This configuration dose not support LE Advertising Extensions functionality APIs.*

*If those functionalities need your system, select BLE Driver (r_ble_extended) configuration.*
*- This configuration dose not support L2CAP functionality APIs.*
*If those functionalities need your system, select BLE Driver (r_ble_extended) configuration.*
*- This configuration dose not support vender specific firmware update functionality APIs.*
*Those APIs are only supported by SPP BLE Abstraction (rm_ble_abs_spp) module.*

### Limitations on FreeRTOS environment

Developers should be aware of the following limitations when using the ble on FreeRTOS environment: When use deep sleep, sleep mode or standby mode, there is following two wake up option from RF module.

- Using Semaphore
- Using Event Groups

*Note*

*Event Groups is processing as a delay handler by timer task.*
*Therefore, if blocked by interrupts or if the timer task is blocked by another high-priority task, the existing connection will be broken.*

## 5.2.12.9 BLE Driver (r_ble_compact)

Modules » Networking

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

# Overview

The Bluetooth Low Energy (BLE) library in Compact configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

### Features

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.
- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.

    - Read/Write characteristic.
- Vendor Specific
    - DTM.
    - Set/Get transmit power.
    - Set/Get BD_ADDR.

### Supported functions

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

| BLE library feature | Extended | Balance | Compact |
|---|---|---|---|
| Common API | Supported | Supported | **Supported** |
| GATT Common API | Supported | *1 Limited | ***1 *2 Limited** |
| GATT Common API | Supported | Supported | **Supported** |
| GATT Server API | Supported | Supported | **Supported** |
| GATT Client API | Supported | Supported | **Supported** |
| L2CAP API | Supported | Not supported | **Not supported** |
| Vendor Specific API | *3 Limited | *3 Limited | ***3 Limited** |

*Note*

*1. This configuration dose not support LE Advertising Extensions functionality APIs.*
*2. This configuration dose not support Central and Observer functionality APIs.*
*3. This configuration dose not support vender specific firmware update functionality APIs.*

### Target Devices

The Renesas BLE Library supports the following devices.

- RA4W1

# Configuration

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

Figure shows the software structure of the BLE FSP module.

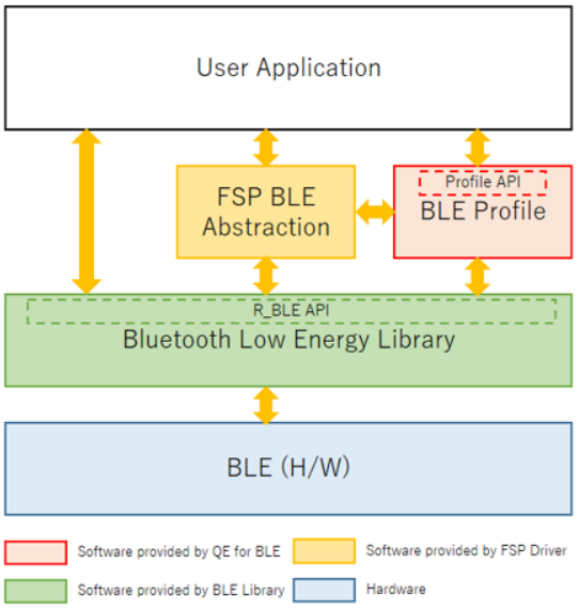



Figure 311: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

### Initialize the BLE protocol stack

*Note*

*It takes around 250msec to initialize BLE protocol stack.*
*R_BLE_Open API will be occupied MCU resources during the initialization.*

### Limitations

Developers should be aware of the following limitations when using the ble:

*Note*

*- This configuration dose not supports LE Advertising Extensions functionality APIs.*
*If those functionalities need your system, select BLE Driver (r_ble_extended) configuration.*
*- This configuration dose not supports Central and Observer functionality APIs.*
*If those functionalities need your system, select BLE Driver (r_ble_extended) or BLE Driver (r_ble_balance) configuration.*
*- This configuration dose not supports L2CAP functionality APIs.*
*If those functionalities need your system, select BLE Driver (r_ble_extended) configuration.*
*- This configuration dose not supports vender specific firmware update functionality APIs.*
*Those APIs are only supported by SPP BLE Abstraction (rm_ble_abs_spp) module.*

### Limitations on FreeRTOS environment

Developers should be aware of the following limitations when using the ble on FreeRTOS environment: When use deep sleep, sleep mode or standby mode, there is following two wake up option from RF module.

- Using Semaphore
- Using Event Groups

*Note*

*Event Groups is processing as a delay handler by timer task.*
*Therefore, if blocked by interrupts or if the timer task is blocked by another high-priority task, the existing connection will be broken.*

### 5.2.12.10 BLE Driver (r_ble_extended)

Modules » Networking

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

# Overview

The Bluetooth Low Energy (BLE) library in Extended configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

**Features**

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising (Support LE Advertising Extensions).
  - Start/Stop Scan.
  - Connect/Disconnect a link.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.
- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.
  - Read/Write characteristic.
- L2CAP
  - Credit-based flow control transaction.
- Vendor Specific
  - DTM.
  - Set/Get transmit power.
  - Set/Get BD_ADDR.

**Supported functions**

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

| BLE library feature | Extended | Balance | Compact |
|---|---|---|---|
| Common API | **Supported** | Supported | Supported |
| GAP API | **Supported** | *1 Limited | *1 *2 Limited |
| GATT Common API | **Supported** | Supported | Supported |
| GATT Server API | **Supported** | Supported | Supported |
| GATT Client API | **Supported** | Supported | Supported |
| L2CAP API | **Supported** | Not supported | Not supported |
| Vendor Specific API | ***3 Limited** | *3 Limited | *3 Limited |

*Note*

*1. This configuration dose not support LE Advertising Extensions functionality APIs.*
*2. This configuration dose not support Central and Observer functionality APIs.*
*3. This configuration dose not support vender specific firmware update functionality APIs.*

### Target Devices

The Renesas BLE Library supports the following devices.

- RA4W1

# Configuration

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

Figure shows the software structure of the BLE FSP module.

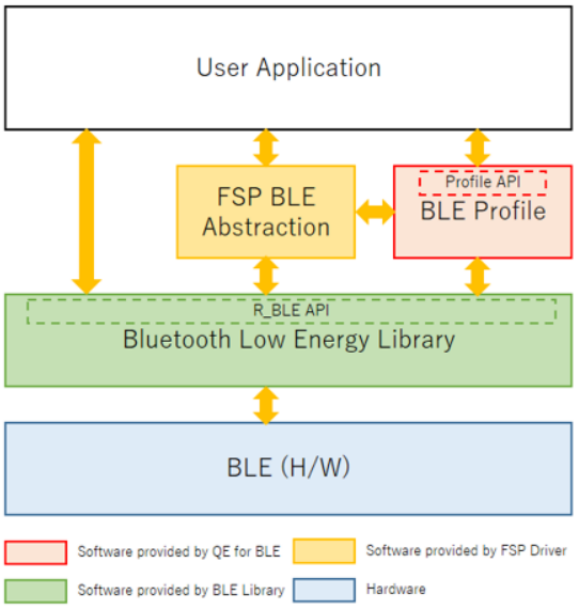



Figure 312: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

### Initialize the BLE protocol stack

*Note*

*It takes around 250msec to initialize BLE protocol stack.*
*R_BLE_Open API will be occupied MCU resources during the initialization.*

### Limitations

Developers should be aware of the following limitations when using the ble:

*Note*

*- This configuration dose not support vender specific firmware update functionality APIs.*
*Those APIs are only supported by SPP BLE Abstraction (rm_ble_abs_spp) module.*

### Limitations on FreeRTOS environment

Developers should be aware of the following limitations when using the ble on FreeRTOS environment: When use deep sleep, sleep mode or standby mode, there is following two wake up option from RF module.

- Using Semaphore
- Using Event Groups

*Note*

*Event Groups is processing as a delay handler by timer task.*
*Therefore, if blocked by interrupts or if the timer task is blocked by another high-priority task, the existing connection will be broken.*

### 5.2.12.11 BLE Mesh Network Modules

Modules » Networking

#### Detailed Description

BLE Mesh Network Modules.

#### Modules

| | |
|---|---|
| | BLE Mesh Network (rm_ble_mesh) |
| | BLE Mesh Network Access (rm_ble_mesh_access) |
| | BLE Mesh Network Bearer (rm_ble_mesh_bearer) |
| | BLE Mesh Network Bearer Platform (rm_mesh_bearer_platform) |
| | BLE Mesh Network Config Client (rm_mesh_config_clt) |
| | BLE Mesh Network Config Server (rm_mesh_config_srv) |
| | BLE Mesh Network Generic Admin Property Server (rm_mesh_generic_admin_prop_srv) |
| | BLE Mesh Network Generic Battery Client (rm_mesh_generic_battery_clt) |
| | BLE Mesh Network Generic Battery Server (rm_mesh_generic_battery_srv) |
| | BLE Mesh Network Generic Client Property Server (rm_mesh_generic_client_prop_srv) |
| | BLE Mesh Network Generic Default Transition Time Client (rm_mesh_generic_dtt_clt) |
| | BLE Mesh Network Generic Default Transition Time Server (rm_mesh_generic_dtt_srv) |
| | BLE Mesh Network Generic Level Client (rm_mesh_generic_level_clt) |
| | BLE Mesh Network Generic Level Server (rm_mesh_generic_level_srv) |
| | BLE Mesh Network Generic Location Client (rm_mesh_generic_loc_clt) |

| | |
|---|---|
| | BLE Mesh Network Generic Location Server (rm_mesh_generic_loc_srv) |
| | BLE Mesh Network Generic Manufacturer Property Server (rm_mesh_generic_mfr_prop_srv) |
| | BLE Mesh Network Generic On Off Client (rm_mesh_generic_on_off_clt) |
| | BLE Mesh Network Generic On Off Server (rm_mesh_generic_on_off_srv) |
| | BLE Mesh Network Generic Power Level Client (rm_mesh_generic_pl_clt) |
| | BLE Mesh Network Generic Power Level Server (rm_mesh_generic_pl_srv) |
| | BLE Mesh Network Generic Power On Off Client (rm_mesh_generic_poo_clt) |
| | BLE Mesh Network Generic Power On Off Server (rm_mesh_generic_poo_srv) |
| | BLE Mesh Network Generic Property Client (rm_mesh_generic_prop_clt) |
| | BLE Mesh Network Generic User Property Server (rm_mesh_generic_user_prop_srv) |
| | BLE Mesh Network Health Client (rm_mesh_health_clt) |
| | BLE Mesh Network Health Server (rm_mesh_health_srv) |
| | BLE Mesh Network Light Control Client (rm_mesh_light_ctl_clt) |
| | BLE Mesh Network Light Control Server (rm_mesh_light_ctl_srv) |
| | BLE Mesh Network Light Hsl Client (rm_mesh_light_hsl_clt) |
| | BLE Mesh Network Light Hsl Server (rm_mesh_light_hsl_srv) |
| | BLE Mesh Network Light Lightness Client (rm_mesh_light_lightness_clt) |
| | BLE Mesh Network Light Lightness Server (rm_mesh_light_lightness_srv) |
| | BLE Mesh Network Light Location Client (rm_mesh_light_lc_clt) |

| | |
|---|---|
| | BLE Mesh Network Light Location Server (rm_mesh_light_lc_srv) |
| | BLE Mesh Network Light Xyl Client (rm_mesh_light_xyl_clt) |
| | BLE Mesh Network Light Xyl Server (rm_mesh_light_xyl_srv) |
| | BLE Mesh Network Lower Trans (rm_ble_mesh_lower_trans) |
| | BLE Mesh Network Network (rm_ble_mesh_network) |
| | BLE Mesh Network Provision (rm_ble_mesh_provision) |
| | BLE Mesh Network Scene Client (rm_mesh_scene_clt) |
| | BLE Mesh Network Scene Server (rm_mesh_scene_srv) |
| | BLE Mesh Network Scheduler Client (rm_mesh_scheduler_clt) |
| | BLE Mesh Network Scheduler Server (rm_mesh_scheduler_srv) |
| | BLE Mesh Network Sensor Client (rm_mesh_sensor_clt) |
| | BLE Mesh Network Sensor Server (rm_mesh_sensor_srv) |
| | BLE Mesh Network Time Client (rm_mesh_time_clt) |
| | BLE Mesh Network Time Server (rm_mesh_time_srv) |
| | BLE Mesh Network Upper Trans (rm_ble_mesh_upper_trans) |

## BLE Mesh Network (rm_ble_mesh)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_Open (rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_Close (rm_ble_mesh_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_StartTransitionTimer (rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_access_state_transition_t const *const p_transition, uint16_t *const p_transition_time_handle) |
| fsp_err_t | RM_BLE_MESH_StopTransitionTimer (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_GetRemainingTransitionTime (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint8_t *const p_remaining_transition_time) |
| fsp_err_t | RM_BLE_MESH_GetRemainingTransitionTimeWithOffset (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint32_t offset_in_ms, uint8_t *const p_remaining_transition_time) |
| fsp_err_t | RM_BLE_MESH_ConvertTransitionTimeFromMs (rm_ble_mesh_ctrl_t *const p_ctrl, uint32_t transition_time_in_ms, uint8_t *const p_transition_time) |
| fsp_err_t | RM_BLE_MESH_ConvertTransitionTimeToMs (rm_ble_mesh_ctrl_t *const p_ctrl, uint8_t transition_time, uint32_t *const p_transition_time_in_ms) |

### Detailed Description

# Overview

Bluetooth Mesh defines a managed-flood-based mesh network. Any device in the network can send a message at any time as long as there is a sufficient density of devices that are listening and relaying messages. See sample application document and start up guide for information on how to create a BLE MESH application.

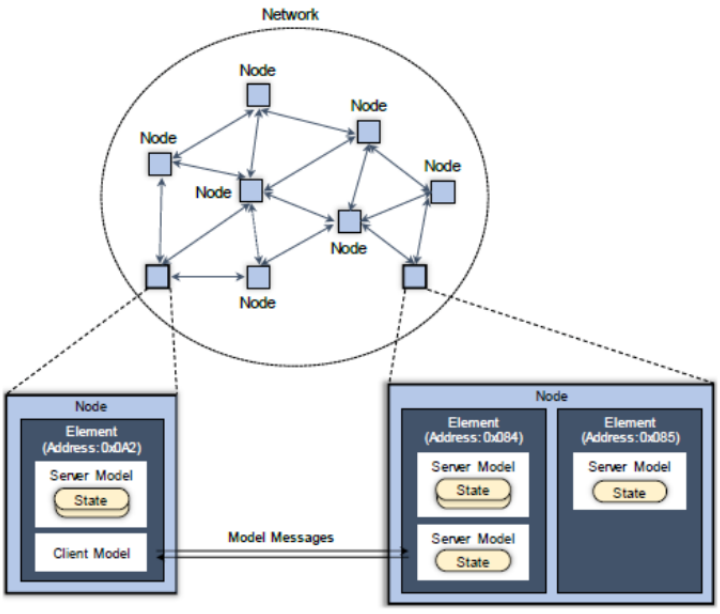



Figure 313: Overview Of Bletooth Mesh

### Features

The BLE Mesh middleware has the following features:

- Supports one-to-one and one-to-many message transmission
- Supports relay messages to other nodes
- Supports secure message transmission against the following attack
    - Wiretap attack
    - Man-in-the-middle attack
    - Replay attack
    - Trash-can attack
    - Brute Force Key attack
- Supports following optional features
    - Supports Relay feature
    - Supports Proxy feature
    - Supports Friend feature
    - Supports Low Power feature

### Target Devices

The BLE Mesh Network module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enable<br>• Disable | Default (BSP) | Specify whether to include code for API parameter checking. Valid settings include. |

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh (rm_ble_mesh)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh (rm_ble_mesh).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Bearer | | | |
| Network Interface Number | Invalid Network Interface Number | 2 | Network interfaces number. |
| Provisioning Interface | Invalid Provisioning | 2 | Provisioning interfaces |

| | | | |
|---|---|---|---|
| Number | Interface Number | | number. |
| Provisioning | | | |
| Unprovisioned Device Beacon Timeout in Milliseconds | Invalid Unprovisioned Device Beacon Timeout in Milliseconds | 200 | Unprovisioned device beacon timeout in milliseconds. |
| Network | | | |
| Network Cache Size | Invalid Network Cache Size | 10 | Network cache size. |
| Network Sequence Number Cache Size | Invalid Network Sequence Number Cache Size | 32 | Network sequence number cache size. |
| Maximum Number of Subnet | Invalid Maximum Number of Subnets | 4 | Maximum number of subnets |
| Maximum Number of Device Key | Invalid Maximum Number of Device Keys | 4 | Maximum number of device keys. |
| Proxy Filter List Size | Invalid Proxy Filter List Size | 2 | Maximum number of addresses present in each proxy filter list. |
| Network Sequence Number Block Size | Invalid Network Sequence Number Block Size | 2048 | The distance between the network sequence numbers, for every persistent storage write. |
| Network Transmit Count for Network Packets | Invalid Network Transmit Count for Network Packets | 1 | Network transmit count for network packets. |
| Network Interval Steps for Network Packets | Invalid Network Interval Steps for Network Packets | 4 | Network interval steps for network packets. |
| Network Transmit Count for Relayed Packets | Invalid Network Transmit Count for Relayed Packets | 0 | Network transmit count for relayed packets. |
| Network Interval Steps for Relayed Packets | Invalid Network Interval Steps for Relayed Packets | 9 | Network interval steps for relayed packets. |
| Proxy ADV Network ID Timeout for Each Subnet in Milliseconds. | Invalid Proxy ADV Network ID Timeout | 100 | Proxy ADV network ID timeout for each subnet in milliseconds. |
| Proxy ADV Node Identity Timeout for Each Subnet in Milliseconds | Invalid Proxy ADV Node Identity Timeout for Each Subnet in Milliseconds | 300 | Proxy ADV node identity timeout for each subnet in milliseconds. |
| Proxy ADV Node Identity Overall Time Period in Seconds | Invalid Proxy ADV Node Identity Overall Time Period in Seconds | 60 | Proxy ADV node identity overall time period in seconds. |

| | | | |
|---|---|---|---|
| Maximum Number of Queued Messages for Transmission | Invalid Maximum Number of Queued Messages for Transmission | 64 | Maximum number of queued messages for transmission. |
| Transport | | | |
| Maximum Number of LPN | Invalid Maximum Number of LPNs | 1 | Maximum number of LPNs. |
| Replay Protection Cache Size | Invalid Replay Protection Cache Size | 10 | Replay protection cache size. |
| Reassembled Cache Size | Invalid Reassembled Cache Size | 8 | Reassembled cach size. |
| Friend Poll Retry Count | Invalid Friend Poll Retry Count | 10 | Friend poll retry count. |
| Maximum Number of Segmentation and Reassembly Context | Invalid Maximum Number of Segmentation and Reassembly Context | 8 | Number of segmentation and reassembly contexts. |
| Lower Transport Segment Transmission Timeout in Milliseconds | Invalid Lower Transport Segment Transmission Timeout in Milliseconds | 300 | Lower transport segment transmission timeout in milliseconds. |
| Lower Transport Segment Re-Transmission Count | Invalid Lower Transport Segment Re-Transmission Count | 2 | Lower transport segment re-transmission count. |
| Lower Transport Acknowledgement Timeout in Milliseconds | Invalid Lower Transport Acknowledgement Timeout in Milliseconds | 200 | Lower transport acknowledgement timeout in milliseconds. |
| Lower Transport Incomplete Timeout in Seconds | Invalid Lower Transport Incomplete Timeout in Seconds | 20 | Lower transport incomplete timeout in seconds. |
| Friendship Receive Window | Invalid Friendship Receive Window | 100 | Friendship receive window. |
| Maximum Number of Friend Message Queue | Invalid Maximum Number of Friend Messages Queue | 15 | Maximum number of messages that the friend is capabale to queue for a single Low Power Node. |
| Maximum Number of Friend Subscription List | Invalid Maximum Number of Friend Subscription List | 8 | Maximum number of subscription addresses that the friend is capable to store for a single Low Power Node. |
| Friend Clear Confirmation Timeout in Milliseconds | Invalid Friend Clear Confirmation Timeout in Milliseconds | 1000 | Friend clear confirmation timeout in milliseconds. |
| Friend Clear Retry | Invalid Friend Clear | 5 | Friend clear retry |

| | | | |
|---|---|---|---|
| Count | Retry Count | | count. |
| Friendship Retry Timeout in Milliseconds | Invalid Friendship Retry Timeout in Milliseconds | 1200 | Friendship retry timeout in milliseconds. |
| Access | | | |
| Maximum Number of Element | Invalid Maximum Number of Element | 4 | Maximum number of elements. |
| Maximum Number of Model | Invalid Maximum Number of Model | 60 | Maximum number of models. |
| Maximum Number of Application | Invalid Maximum Number of Application | 8 | Maximum number of applications (keys) the device can store information about. |
| Maximum Number of Virtual Address | Invalid Maximum Number of Virtual Address | 8 | Maximum number of virtual addresses the device can store information about. |
| Maximum Number of Non-Virtual Address | Invalid Maximum Number of Non-Virtual Address | 8 | Maximum number of non-virtual addresses the device can store information about. |
| Maximum Number of Transition Timers | Invalid Maximum Number of Transition Timers | 5 | Maximum number of transition timers. |
| Maximum Number of Periodic Step Timers | Invalid Maximum Number of Periodic Step Timers | 5 | Maximum number of periodic step timers. |
| Foundation | | | |
| Config Server Secure Network Beacon Interval | Invalid Config Server Secure Network Beacon Interval | 10 | Config server secure network beacon interval. |
| Maximum Number of Health Server Instance | Invalid Maximum Number of Health Server Instance | 2 | Maximum number of health server instances. |
| Model | | | |
| Maximum Number of Light Lightness Controller Server Instance | Invalid Maximum Number of Light Lightness Controller Server Instance | 1 | Maximum number of light lightness controller server instances. |
| ID | | | |
| Company ID | Invalid Company ID | 0x0036 | Company ID. |
| Product ID | Invalid Product ID | 0x0001 | Product ID. |
| Vendor ID | Invalid Vendor ID | 0x0100 | Vendor ID. |
| Platform | | | |

| | | | |
|---|---|---|---|
| Platform > Storage | | | |
| Block Number | Invalid Block Number | 5 | Block number. |
| Platform > Memory Pool | | | |
| Memory Pool Size | Invalid Memory Pool Size | 0x4000 | Memory pool size. |
| Platform > Logging | | | |
| Packet Bitfield | • Network<br>• Lower Trans<br>• Upper Trans<br>• Access | | Specifies if this is included in the packet_bitfield mask. |
| Module Info Bitfield | • Network<br>• Lower Trans<br>• Upper Trans<br>• Access | | Specifies if this is included in the module information bitfield mask. |
| Generic Log Bitfield | • Network<br>• Lower Trans<br>• Upper Trans<br>• Access | | Specifies if this is included in the generic log bitfield mask. |
| function | Name Must Be a Valid C Symbol | NULL | |

### Clock Configuration

*Note*

*System clock (ICLK): 8 MHz or more*
*Peripheral module clock A (PCLKA): 8MHz or more*
*The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.*
*It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.*

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Mesh System Architecture

Purpose of each layer is,

### Model layer

Model is a standardized typical functionality so that nodes perform operations in accordance with application scenario.

### Foundation Model layer

Foundation Models are models to configure and manage operations of elements.

### Access layer

The access layer defines how higher layer applications can use the upper transport layer.

### Upper transport layer

The upper transport layer encrypts, decrypts, and authenticates application data and is designed to provide confidentiality of access messages.

### Lower transport layer

The lower transport layer defines how upper transport layer messages are segmented and reassembled into multiple Lower Transport PDUs to deliver large upper transport layer messages to other nodes.

### Network layer

The network layer defines how transport messages are addressed towards one or more elements.

### Bearer layer

The bearer layer defines how network messages are transported between nodes.

### Device Life Cycle

The device not joined in the mesh network is an Unprovisioned Device and the device joined in the mesh network is called a Node.

- Unprovisioned Device
  Unprovisioned device cannot send or receive mesh messages. However, it will advertise its presence to the provisioner. By the provisioner, Unprovisioned Devices are invited to join the mesh network and become nodes(Provisioning).
- Node
  Nodes can send and receive mesh messages. It is managed by the configuration client through the mesh network, which configures how the nodes communicate. The configuration client can also remove a Node from the mesh network, which will return the node to an Unprovisioned Device.

To communicate with other nodes by using Models, each node needs Configuration. By Configuration process, information required for Model operation such as Application Keys, Publish Address, Subscription Address is configured. Following shows a typical lifecycle of a node.

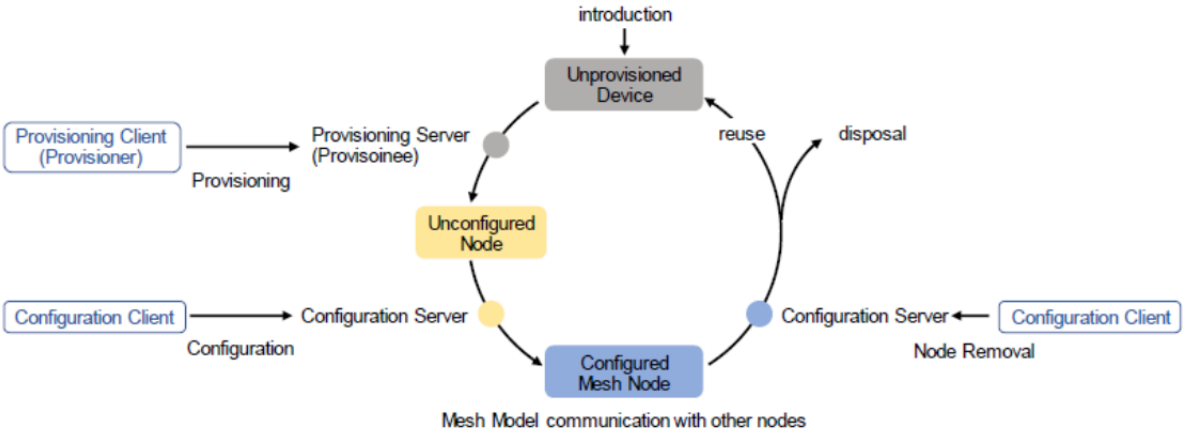



Figure 314: Device Life Cycle Image

Newly introduced device is provisioned by Provisioner and joins a network. Furthermore, this device is configured by Configuration Client and becomes to be able to communicate with other nodes with Mesh Model. Generally, Configuration Client is a smart phone or other mobile computing device. Configuration Client removes a node from a network by sending Config Node Reset message. Besides, Configuration Client updates encryption keys used in the network, and the removed node becomes unable to communicate with other nodes.

# Examples

### BLE_MESH Basic Example

This is a basic example of minimal use of the BLE_MESH in an application.

# License and Copyright Notice

Mesh library includes crackle. The license and copyright of crackle are as follows.

```
BSD 2-Clause License
Copyright (c) 2013-2018, Mike Ryan
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

```
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

## Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_instance_ctrl_t

| struct rm_ble_mesh_instance_ctrl_t |
|---|
| RM_BLE_MESH_BEARER private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_BEARER_Open() is called. |

## Function Documentation

### ◆ RM_BLE_MESH_Open()

| fsp_err_t RM_BLE_MESH_Open ( rm_ble_mesh_ctrl_t *const *p_ctrl*, rm_ble_mesh_cfg_t const *const *p_cfg* ) |
|---|
| Open ble mesh middleware. API to initialize Mesh Stack. This is the first API that the application should call before any other API. This function initializes all the internal stack modules and data structures.<br><br>Implements rm_ble_mesh_api_t::open.<br><br>Example:<br>`/* Open the module. */`<br>`err = RM_BLE_MESH_Open(&g_ble_mesh0_ctrl, &g_ble_mesh0_cfg);`<br><br>**Return values**<br>FSP_SUCCESS — Module opened successfully.<br>FSP_ERR_ASSERTION — Pointer to control block or configuration structure is NULL.<br>FSP_ERR_ALREADY_OPEN — Module is already open. |

### ◆ RM_BLE_MESH_Close()

fsp_err_t RM_BLE_MESH_Close ( rm_ble_mesh_ctrl_t *const *p_ctrl*)

Close ble mesh middleware. API to turn off Bluetooth Hardware. This API should be called after RM_BLE_MESH_Open.

Implements rm_ble_mesh_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_Close(&g_ble_mesh0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_StartTransitionTimer()

fsp_err_t RM_BLE_MESH_StartTransitionTimer ( rm_ble_mesh_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_state_transition_t const *const *p_transition*, uint16_t *const *p_transition_time_handle* )

To start transition timer. API to start a transition timer.

Implements rm_ble_mesh_api_t::startTransitionTimer.

Example:

```
/* Start transition timer. */
   err = RM_BLE_MESH_StartTransitionTimer(&g_ble_mesh0_ctrl, &transition,
&transition_time_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_transition is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |

### ◆ RM_BLE_MESH_StopTransitionTimer()

fsp_err_t RM_BLE_MESH_StopTransitionTimer ( rm_ble_mesh_ctrl_t *const *p_ctrl*, uint16_t *transition_time_handle* )

To stop transition timer. API to stop a transition timer.

Implements rm_ble_mesh_api_t::stopTransitionTimer.

Example:

```
/* Stop transition timer. */
   err = RM_BLE_MESH_StopTransitionTimer(&g_ble_mesh0_ctrl, transition_time_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_GetRemainingTransitionTime()

fsp_err_t RM_BLE_MESH_GetRemainingTransitionTime ( rm_ble_mesh_ctrl_t *const *p_ctrl*, uint16_t *transition_time_handle*, uint8_t *const *p_remaining_transition_time* )

To get remaining Transition Time. API to get remaining Transition Time.

Implements rm_ble_mesh_api_t::getRemainingTransitionTime.

Example:

```
/* Get remaining transition time. */
   err = RM_BLE_MESH_GetRemainingTransitionTime(&g_ble_mesh0_ctrl,
transition_time_handle, &remaining_transition_time);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_remaining_transition_time is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_GetRemainingTransitionTimeWithOffset()

fsp_err_t RM_BLE_MESH_GetRemainingTransitionTimeWithOffset ( rm_ble_mesh_ctrl_t *const *p_ctrl*, uint16_t *transition_time_handle*, uint32_t *offset_in_ms*, uint8_t *const *p_remaining_transition_time* )

To get remaining Transition Time, with offset. API to get remaining Transition Time with offset in ms.

Implements rm_ble_mesh_api_t::getRemainingTransitionTimeWithOffset.

Example:

```
/* Get remaining transition time with offset. */
   err = RM_BLE_MESH_GetRemainingTransitionTimeWithOffset(&g_ble_mesh0_ctrl,
                                                          transition_time_handle,
                                                          offset_in_ms,

&remaining_transition_time);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_remaining_transition_time is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ConvertTransitionTimeFromMs()

fsp_err_t RM_BLE_MESH_ConvertTransitionTimeFromMs ( rm_ble_mesh_ctrl_t *const *p_ctrl*, uint32_t *transition_time_in_ms*, uint8_t *const *p_transition_time* )

To convert transition time from millisecond. API to convert transition timer in millisecond to Generic Default Transition Time state format.

Implements rm_ble_mesh_api_t::convertTransitionTimeFromMs.

Example:

```
/* Convert transition time from milliseconds. */
   err = RM_BLE_MESH_ConvertTransitionTimeFromMs(&g_ble_mesh0_ctrl,
transition_time_in_ms, &transition_time);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_transition_time is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ConvertTransitionTimeToMs()

fsp_err_t RM_BLE_MESH_ConvertTransitionTimeToMs ( rm_ble_mesh_ctrl_t *const *p_ctrl*, uint8_t *transition_time*, uint32_t *const *p_transition_time_in_ms* )

To convert transition time to millisecond. API to convert Generic Default Transition Time state format to required transition time value in miliseconds.

Implements rm_ble_mesh_api_t::convertTransitionTimeToMs.

Example:

```
/* Convert transition time to milliseconds. */
   err = RM_BLE_MESH_ConvertTransitionTimeToMs(&g_ble_mesh0_ctrl, transition_time,
&transition_time_in_ms);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_transition_time_in_ms is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Access (rm_ble_mesh_access)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_ACCESS_Open (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_ACCESS_Close (rm_ble_mesh_access_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_ACCESS_RegisterModel (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_t const *const p_model, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetElementHandle (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t elem_addr, rm_ble_mesh_access_element_handle_t *const p_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetElementHandleForModelHandle (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_element_handle_t *const p_elem_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetModelHandle (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_element_handle_t elem_handle, rm_ble_mesh_access_model_id_t model_id, rm_ble_mesh_access_model_handle_t *const p_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_Publish (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint8_t reliable) |
| fsp_err_t | RM_BLE_MESH_ACCESS_ReliablePublish (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint32_t rsp_opcode) |
| fsp_err_t | RM_BLE_MESH_ACCESS_Reply (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, uint8_t ttl, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw) |
| fsp_err_t | RM_BLE_MESH_ACCESS_ReplyAndPublish (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw, rm_ble_mesh_access_publish_setting_t |

| | |
|---|---|
| | const *const p_publish_setting) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SendPdu (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_pdu_t const *const p_pdu, uint8_t reliable) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetCompositionData (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_buffer_t *const p_buffer) |
| fsp_err_t | RM_BLE_MESH_ACCESS_Reset (rm_ble_mesh_access_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetElementCount (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_count) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetPrimaryUnicastAddress (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetPrimaryUnicastAddress (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t *const p_addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetDefaultTtl (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ttl) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetDefaultTtl (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_ttl) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetIvIndex (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t iv_index, uint8_t iv_update_flag) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetIvIndex (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t *const p_iv_index, uint8_t *const p_iv_update_flag) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetIvIndexByIvi (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ivi, uint32_t *const p_iv_index) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetFeaturesField (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t enable, uint8_t feature) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetFeaturesField (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_enable, uint8_t feature) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetFeatures (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_features) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_ACCESS_GetFriendshipRole (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_friend_role) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetFriendshipRole (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t friend_role) |
| fsp_err_t | RM_BLE_MESH_ACCESS_AddDeviceKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_dev_key, rm_ble_mesh_network_address_t uaddr, uint8_t num_elements) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetDeviceKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t dev_key_index, uint8_t **const p_dev_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_RemoveAllDeviceKeys (rm_ble_mesh_access_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetProvisionedDeviceList (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_provisioned_device_entry_t const *const p_prov_dev_list, uint16_t *const p_num_entries) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetDeviceKeyHandle (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t prim_elem_uaddr, rm_ble_mesh_access_device_key_handle_t *const p_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_appkey_handle_t appkey_handle, uint8_t **const p_app_key, uint8_t *const p_aid) |
| fsp_err_t | RM_BLE_MESH_ACCESS_AddUpdateNetkey (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, uint32_t opcode, uint8_t const *const p_net_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_AddFriendSecurityCredential (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index, rm_ble_mesh_access_friend_security_credential_info_t info) |
| fsp_err_t | RM_BLE_MESH_ACCESS_DeleteFriendSecurityCredential (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index) |
| fsp_err_t | RM_BLE_MESH_ACCESS_FindSubnet (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_FindMasterSubnet (rm_ble_mesh_access_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_network_subnet_handle_t friend_subnet_handle, rm_ble_mesh_network_subnet_handle_t *const p_master_subnet_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_DeleteNetKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetNetKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_net_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetNetKeyIndexList (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_netkey_count, uint16_t *const p_netkey_index_list) |
| fsp_err_t | RM_BLE_MESH_ACCESS_LookUpNid (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t nid, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set) |
| fsp_err_t | RM_BLE_MESH_ACCESS_LookUpNetworkId (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_network_id, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set) |
| fsp_err_t | RM_BLE_MESH_ACCESS_LookUpAid (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t aid, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle, uint8_t *const p_app_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetProvisioningData (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_provision_data_t const *const p_prov_data) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetNid (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_nid) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetPrivacyKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_privacy_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetNetworkId (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_network_id) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetBeaconKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const |

| | |
|---|---|
| | p_beacon_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetIdentityKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_identity_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetSubnetEncryptionKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_encrypt_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetNodeIdentity (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetNodeIdentity (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetKeyRefreshPhase (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_key_refresh_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetKeyRefreshPhase (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t const *const p_key_refresh_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetTransmitState (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t tx_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetTransmitState (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t *const p_tx_state) |
| fsp_err_t | RM_BLE_MESH_ACCESS_AddAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_UpdateAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_DeleteAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetAppKeyHandle (rm_ble_mesh_access_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetAppKeyIndexList (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list) |
| fsp_err_t | RM_BLE_MESH_ACCESS_BindModelWithAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index) |
| fsp_err_t | RM_BLE_MESH_ACCESS_UnbindModelWithAppKey (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetModelAppKeyList (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetModelPublication (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info) |
| fsp_err_t | RM_BLE_MESH_ACCESS_SetModelPublicationPeriodDivisor (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint8_t period_divisor) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetModelPublication (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info) |
| fsp_err_t | RM_BLE_MESH_ACCESS_AddModelSubscription (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_DeleteModelSubscription (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_DeleteAllModelSubscription (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_ACCESS_GetModelSubscriptionList (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list) |
| fsp_err_t | RM_BLE_MESH_ACCESS_GetAllModelSubscriptionList (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list) |
| fsp_err_t | RM_BLE_MESH_ACCESS_IsValidElementAddress (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_IsFixedGroupAddressToBeProcessed (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_IsValidSubscriptionAddress (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| fsp_err_t | RM_BLE_MESH_ACCESS_EnableIvUpdateTestMode (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_iv_update_test_mode_t mode) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Access module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_access

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_access_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Access (rm_ble_mesh_access)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Access (rm_ble_mesh_access).

| Configuration | Options | Default | Description |
|---|---|---|---|

General

| | | | |
|---|---|---|---|
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_access0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Location Descriptor | Invalid Descriptor | 0 | Location descriptor. |
| Element Number | Invalid Descriptor | 0 | Element number to register the model. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_access_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_access_instance_ctrl_t

| struct rm_ble_mesh_access_instance_ctrl_t |
|---|
| RM_BLE_MESH_ACCESS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_ACCESS_Open() is called. |

## Function Documentation

### ◆ RM_BLE_MESH_ACCESS_Open()

fsp_err_t RM_BLE_MESH_ACCESS_Open ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_cfg_t const *const *p_cfg* )

Open access middleware. This routine creates a new node in the device. This can be used by the application to create extra nodes if required in addition to the default primary node. And this routine registers an element that can be populated with the models information to a specific node in the device identified by the node id.

Implements rm_ble_mesh_access_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_ACCESS_Open(&g_ble_mesh_access0_ctrl, &g_ble_mesh_access0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_Close()

fsp_err_t RM_BLE_MESH_ACCESS_Close ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*)

Close access middleware. Implements rm_ble_mesh_access_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_ACCESS_Close(&g_ble_mesh_access0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_ACCESS_RegisterModel()

fsp_err_t RM_BLE_MESH_ACCESS_RegisterModel ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_t const *const *p_model*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

Register a model with the access layer. This routine registers a model associated with an element with the access layer.

Implements rm_ble_mesh_access_api_t::registerModel.

Example:

```
/* Register a model with the access layer. */
   err = RM_BLE_MESH_ACCESS_RegisterModel(&g_ble_mesh_access0_ctrl, &model,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_model and p_model_handle are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetElementHandle()

fsp_err_t RM_BLE_MESH_ACCESS_GetElementHandle ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *elem_addr*, rm_ble_mesh_access_element_handle_t *const *p_handle* )

Get element handle. This routine searches for the element handle associated with specific element address.

Implements rm_ble_mesh_access_api_t::getElementHandle.

Example:

```
/* Get element handle. */
   err = RM_BLE_MESH_ACCESS_GetElementHandle(&g_ble_mesh_access0_ctrl, elem_addr,
&elem_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |

### ◆ RM_BLE_MESH_ACCESS_GetElementHandleForModelHandle()

fsp_err_t RM_BLE_MESH_ACCESS_GetElementHandleForModelHandle ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, rm_ble_mesh_access_element_handle_t *const *p_elem_handle* )

Get element handle for a given model handle. This routine searches for the element handle associated with specific model handle.

Implements rm_ble_mesh_access_api_t::getElementHandleForModelHandle.

Example:

```
/* Get element handle for a given model handle. */
   err = RM_BLE_MESH_ACCESS_GetElementHandleForModelHandle(&g_ble_mesh_access0_ctrl,
model_handle, &elem_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_elem_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetModelHandle()

fsp_err_t RM_BLE_MESH_ACCESS_GetModelHandle ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_element_handle_t *elem_handle*, rm_ble_mesh_access_model_id_t *model_id*, rm_ble_mesh_access_model_handle_t *const *p_handle* )

Get model handle. This routine searches for the model handle associated with specific model ID.

Implements rm_ble_mesh_access_api_t::getModelHandle.

Example:

```
/* Get model handle. */
   err = RM_BLE_MESH_ACCESS_GetModelHandle(&g_ble_mesh_access0_ctrl, elem_handle,
model_id, &model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_Publish()

fsp_err_t RM_BLE_MESH_ACCESS_Publish ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t const *const *p_handle*, rm_ble_mesh_access_req_msg_raw_t const *const *p_publish_message*, uint8_t *reliable* )

API to publish access layer message. This routine publishes Access Layer message to the publish address associated with the model.

Implements rm_ble_mesh_access_api_t::publish.

Example:

```
/* Publish access layer message. */
   err = RM_BLE_MESH_ACCESS_Publish(&g_ble_mesh_access0_ctrl, &model_handle,
&publish_message, reliable);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle and p_publish_message are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_ACCESS_ReliablePublish()

fsp_err_t RM_BLE_MESH_ACCESS_ReliablePublish ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t const *const *p_handle*, rm_ble_mesh_access_req_msg_raw_t const *const *p_publish_message*, uint32_t *rsp_opcode* )

API to reliably publish Access layer message. This routine reliably publishes Access Layer message to the publish address associated with the model.

Implements rm_ble_mesh_access_api_t::reliablePublish.

Example:

```
/* Reliably publish access layer message. */
   err = RM_BLE_MESH_ACCESS_ReliablePublish(&g_ble_mesh_access0_ctrl, &model_handle,
&req_msg_raw, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle and p_publish_message are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_ACCESS_Reply()

fsp_err_t RM_BLE_MESH_ACCESS_Reply ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_req_msg_context_t const *const *p_req_msg_context*, uint8_t *ttl*, rm_ble_mesh_access_req_msg_raw_t const *const *p_req_msg_raw* )

API to reply to Access Layer message. This routine replies to Access Layer message.

Implements rm_ble_mesh_access_api_t::reply.

Example:

```
/* Reply to access layer message. */
   err = RM_BLE_MESH_ACCESS_Reply(&g_ble_mesh_access0_ctrl, &req_msg_context, ttl,
&req_msg_raw);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_req_msg_context and p_req_msg_raw are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_ACCESS_ReplyAndPublish()

fsp_err_t RM_BLE_MESH_ACCESS_ReplyAndPublish ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_req_msg_context_t const *const *p_req_msg_context*, rm_ble_mesh_access_req_msg_raw_t const *const *p_req_msg_raw*, rm_ble_mesh_access_publish_setting_t const *const *p_publish_setting* )

API to reply to Access Layer message and optionally also to publish. This routine replies to Access Layer message and also publish if requested by application.

Implements rm_ble_mesh_access_api_t::replyAndPublish.

Example:

```
/* Reply to access layer message. */
   err =
RM_BLE_MESH_ACCESS_ReplyAndPublish(&g_ble_mesh_access0_ctrl, &req_msg_context,
&req_msg_raw, &publish_setting);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_req_msg_context and p_req_msg_raw are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_ACCESS_SendPdu()

fsp_err_t RM_BLE_MESH_ACCESS_SendPdu ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_pdu_t const *const *p_pdu*, uint8_t *reliable* )

API to send access PDUs. This routine sends transport PDUs to peer device.

Implements rm_ble_mesh_access_api_t::sendPdu.

Example:

```
/* Send Access PDUs. */
   err = RM_BLE_MESH_ACCESS_SendPdu(&g_ble_mesh_access0_ctrl, &pdu, reliable);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_pdu is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetCompositionData()

fsp_err_t RM_BLE_MESH_ACCESS_GetCompositionData ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_buffer_t *const *p_buffer* )

Get composition data.

Implements rm_ble_mesh_access_api_t::getCompositionData.

Example:

```
/* Get composition data. */
   err = RM_BLE_MESH_ACCESS_GetCompositionData(&g_ble_mesh_access0_ctrl, &buffer);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_buffer is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_Reset()

fsp_err_t RM_BLE_MESH_ACCESS_Reset ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*)

To reset a node. This routine resets a node (other than a Provisioner) and removes it from the network.

Implements rm_ble_mesh_access_api_t::reset.

Example:

```
/* Reset a node. */
   err = RM_BLE_MESH_ACCESS_Reset(&g_ble_mesh_access0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetElementCount()

fsp_err_t RM_BLE_MESH_ACCESS_GetElementCount ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *const *p_count* )

To get the number of elements in local node. This routine retrieves the number of elements in local node.

Implements rm_ble_mesh_access_api_t::getElementCount.

Example:

```
/* Get the number of elements in local node. */
   err = RM_BLE_MESH_ACCESS_GetElementCount(&g_ble_mesh_access0_ctrl, &count);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_count is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_SetPrimaryUnicastAddress()

fsp_err_t RM_BLE_MESH_ACCESS_SetPrimaryUnicastAddress ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr* )

To set primary unicast address. This routine sets primary unicast address.

Implements rm_ble_mesh_access_api_t::setPrimaryUnicastAddress.

Example:

```
/* Set primary unicast address. */
   err = RM_BLE_MESH_ACCESS_SetPrimaryUnicastAddress(&g_ble_mesh_access0_ctrl,
addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |

### ◆ RM_BLE_MESH_ACCESS_GetPrimaryUnicastAddress()

fsp_err_t RM_BLE_MESH_ACCESS_GetPrimaryUnicastAddress ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *const *p_addr* )

To get primary unicast address. This routine gets primary unicast address.

Implements rm_ble_mesh_access_api_t::getPrimaryUnicastAddress.

Example:

```
/* Get primary unicast address. */
   err = RM_BLE_MESH_ACCESS_GetPrimaryUnicastAddress(&g_ble_mesh_access0_ctrl,
&addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_addr is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_SetDefaultTtl()

fsp_err_t RM_BLE_MESH_ACCESS_SetDefaultTtl ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *ttl* )

To set default TTL. This routine sets default TTL.

Implements rm_ble_mesh_access_api_t::setDefaultTtl.

Example:

```
/* Set default TTL. */
   err = RM_BLE_MESH_ACCESS_SetDefaultTtl(&g_ble_mesh_access0_ctrl, ttl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetDefaultTtl()

fsp_err_t RM_BLE_MESH_ACCESS_GetDefaultTtl ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t const *const *p_ttl* )

To get default TTL. This routine gets default TTL.

Implements rm_ble_mesh_access_api_t::getDefaultTtl.

Example:

```
/* Get default TTL. */
   err = RM_BLE_MESH_ACCESS_GetDefaultTtl(&g_ble_mesh_access0_ctrl, &ttl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_ttl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_SetIvIndex()

fsp_err_t RM_BLE_MESH_ACCESS_SetIvIndex ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint32_t *iv_index*, uint8_t *iv_update_flag* )

To set IV Index. This routine sets IV Index.

Implements rm_ble_mesh_access_api_t::setIvIndex.

Example:

```
/* Set IV Index. */
   err = RM_BLE_MESH_ACCESS_SetIvIndex(&g_ble_mesh_access0_ctrl, iv_index,
iv_update_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |
| FSP_ERR_IN_USE | Resource is busy. |

### ◆ RM_BLE_MESH_ACCESS_GetIvIndex()

fsp_err_t RM_BLE_MESH_ACCESS_GetIvIndex ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint32_t *const *p_iv_index*, uint8_t *const *p_iv_update_flag* )

To get IV Index. This routine gets IV Index.

Implements rm_ble_mesh_access_api_t::getIvIndex.

Example:

```
/* Get IV Index. */
   err = RM_BLE_MESH_ACCESS_GetIvIndex(&g_ble_mesh_access0_ctrl, &iv_index,
&iv_update_flag);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_iv_index and p_iv_update_flag are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetIvIndexByIvi()

fsp_err_t RM_BLE_MESH_ACCESS_GetIvIndexByIvi ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *ivi*, uint32_t *const *p_iv_index* )

To get IV Index by IVI. This routine gets IV Index based on the IVI in the received packet.

Implements rm_ble_mesh_access_api_t::getIvIndexByIvi.

Example:

```
/* Get IV Index by IVI. */
   err = RM_BLE_MESH_ACCESS_GetIvIndexByIvi(&g_ble_mesh_access0_ctrl, ivi,
&iv_index);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_iv_index is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_SetFeaturesField()

fsp_err_t RM_BLE_MESH_ACCESS_SetFeaturesField ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *enable*, uint8_t *feature* )

To enable/disable a feature. This routine enables/disables a feature field.

Implements rm_ble_mesh_access_api_t::setFeaturesField.

Example:

```
/* Enable/Disable a feature. */
   err = RM_BLE_MESH_ACCESS_SetFeaturesField(&g_ble_mesh_access0_ctrl, enable,
features);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetFeaturesField()

fsp_err_t RM_BLE_MESH_ACCESS_GetFeaturesField ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *const *p_enable*, uint8_t *feature* )

To get state of a feature. This routine gets the state of a feature field.

Implements rm_ble_mesh_access_api_t::getFeaturesField.

Example:

```
/* Get state of a feature. */
   err = RM_BLE_MESH_ACCESS_GetFeaturesField(&g_ble_mesh_access0_ctrl, &enable,
features);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_enable is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetFeatures()

fsp_err_t RM_BLE_MESH_ACCESS_GetFeatures ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *const *p_features* )

To get state of all features. This routine gets the state of all features.

Implements rm_ble_mesh_access_api_t::getFeatures.

Example:

```
/* Get state of all features. */
   err = RM_BLE_MESH_ACCESS_GetFeatures(&g_ble_mesh_access0_ctrl, &features);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_features is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetFriendshipRole()

fsp_err_t RM_BLE_MESH_ACCESS_GetFriendshipRole ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *const *p_friend_role* )

To get friendship role of the node. This routine gets the current friendship role of the node.

Implements rm_ble_mesh_access_api_t::getFriendshipRole.

Example:

```
/* Get friendship role of the node. */
   err = RM_BLE_MESH_ACCESS_GetFriendshipRole(&g_ble_mesh_access0_ctrl,
&friend_role);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_friend_role is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_SetFriendshipRole()

fsp_err_t RM_BLE_MESH_ACCESS_SetFriendshipRole ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *friend_role* )

To set friendship role of the node. This routine sets the current friendship role of the node.

Implements rm_ble_mesh_access_api_t::setFriendshipRole.

Example:

```
/* Set friendship role of the node. */
   err = RM_BLE_MESH_ACCESS_SetFriendshipRole(&g_ble_mesh_access0_ctrl,
friend_role);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_AddDeviceKey()

fsp_err_t RM_BLE_MESH_ACCESS_AddDeviceKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t const *const *p_dev_key*, rm_ble_mesh_network_address_t *uaddr*, uint8_t *num_elements* )

To add Device Key. This routine adds Device Key entry, along with corresponding Primary Device Address and Number of elements.

Implements rm_ble_mesh_access_api_t::addDeviceKey.

Example:

```
/* Add Device Key. */
   err = RM_BLE_MESH_ACCESS_AddDeviceKey(&g_ble_mesh_access0_ctrl, &dev_key, uaddr,
num_elements);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_dev_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OVERFLOW | Device key table is full. |

### ◆ RM_BLE_MESH_ACCESS_GetDeviceKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetDeviceKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *dev_key_index*, uint8_t **const *p_dev_key* )

To get Device Key. This routine gets Device Key entry.

Implements rm_ble_mesh_access_api_t::getDeviceKey.

Example:

```
/* Get Device Key. */
   err = RM_BLE_MESH_ACCESS_GetDeviceKey(&g_ble_mesh_access0_ctrl, dev_key_index,
p_dev_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_dev_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_RemoveAllDeviceKeys()

fsp_err_t RM_BLE_MESH_ACCESS_RemoveAllDeviceKeys ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*)

To remove all Device Keys. This routine removes all Device Keys from table.

Implements rm_ble_mesh_access_api_t::removeAllDeviceKeys.

Example:

```
/* Remove all Device Keys. */
   err = RM_BLE_MESH_ACCESS_RemoveAllDeviceKeys(&g_ble_mesh_access0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetProvisionedDeviceList()

fsp_err_t RM_BLE_MESH_ACCESS_GetProvisionedDeviceList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_provisioned_device_entry_t const *const *p_prov_dev_list*, uint16_t *const *p_num_entries* )

To get list of Provisioned Device List. This routine returns list of Provisioned Devices from the Device Key Table.

Implements rm_ble_mesh_access_api_t::getProvisionedDeviceList.

Example:

```
/* Get list of provisioned device list. */
   err = RM_BLE_MESH_ACCESS_GetProvisionedDeviceList(&g_ble_mesh_access0_ctrl,
&prov_dev_list, &num_entries);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_prov_dev_list and p_num_entries are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetDeviceKeyHandle()

fsp_err_t RM_BLE_MESH_ACCESS_GetDeviceKeyHandle ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *prim_elem_uaddr*, rm_ble_mesh_access_device_key_handle_t *const *p_handle* )

To get device key handle. This routine returns Device Key Handle for a given Primary Element Address entry in Device Key Table.

Implements rm_ble_mesh_access_api_t::getDeviceKeyHandle.

Example:

```
/* Get Device Key handle. */
   err = RM_BLE_MESH_ACCESS_GetDeviceKeyHandle(&g_ble_mesh_access0_ctrl,
prim_elem_uaddr, &handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_appkey_handle_t *appkey_handle*, uint8_t **const *p_app_key*, uint8_t *const *p_aid* )

To get AppKey. This routine gets AppKey along with AID entry.

Implements rm_ble_mesh_access_api_t::getAppKey.

Example:

```
/* Get AppKey. */
   err = RM_BLE_MESH_ACCESS_GetAppKey(&g_ble_mesh_access0_ctrl, appkey_handle,
p_app_key, &aid);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_app_key and p_aid are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_AddUpdateNetkey()

fsp_err_t RM_BLE_MESH_ACCESS_AddUpdateNetkey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint16_t *netkey_index*, uint32_t *opcode*, uint8_t const *const *p_net_key* )

To add/update NetKey. This routine adds/updates NetKey entry. Each NetKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::addUpdateNetkey.

Example:

```
/* Add/Update NetKey. */
   err = RM_BLE_MESH_ACCESS_AddUpdateNetkey(&g_ble_mesh_access0_ctrl, netkey_index,
opcode, &net_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_net_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_AddFriendSecurityCredential()

fsp_err_t RM_BLE_MESH_ACCESS_AddFriendSecurityCredential ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *friend_index*, rm_ble_mesh_access_friend_security_credential_info_t *info* )

To add Security Credential of a LPN or the Friend. This routine adds NID, privacy and encryption keys associated with a friendship.

Implements rm_ble_mesh_access_api_t::addFriendSecurityCredential.

Example:

```
/* Add security credential of a LPN or the friend. */
   err =
RM_BLE_MESH_ACCESS_AddFriendSecurityCredential(&g_ble_mesh_access0_ctrl,
subnet_handle, friend_index, info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_DeleteFriendSecurityCredential()

fsp_err_t RM_BLE_MESH_ACCESS_DeleteFriendSecurityCredential ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *friend_index* )

To delete the Security Credential of a LPN or the Friend. This routine deletes NID, privacy and encryption keys associated with a friendship.

Implements rm_ble_mesh_access_api_t::deleteFriendSecurityCredential.

Example:

```
/* Delete the security credential of a LPN or the Friend. */
   err = RM_BLE_MESH_ACCESS_DeleteFriendSecurityCredential(&g_ble_mesh_access0_ctrl,
subnet_handle, friend_index);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_FindSubnet()

fsp_err_t RM_BLE_MESH_ACCESS_FindSubnet ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint16_t *netkey_index*, rm_ble_mesh_network_subnet_handle_t *const *p_subnet_handle* )

To find a Subnet associated with the NetKey. This routine finds a Subnet based on the NetKey entry. Each NetKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::findSubnet.

Example:

```
/* Find a subnet associated with the NetKey. */
   err = RM_BLE_MESH_ACCESS_FindSubnet(&g_ble_mesh_access0_ctrl, netkey_index,
&subnet_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_subnet_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_FindMasterSubnet()

fsp_err_t RM_BLE_MESH_ACCESS_FindMasterSubnet ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *friend_subnet_handle*, rm_ble_mesh_network_subnet_handle_t *const* *p_master_subnet_handle* )

To find the Master Subnet associated with the friend security credential, identified by Friend Subnet Handle. This routine finds the Master Subnet based on the friend security credential, identified by Friend Subnet Handle.

Implements rm_ble_mesh_access_api_t::findMasterSubnet.

Example:

```
/* Close the module. */
   err =
RM_BLE_MESH_ACCESS_FindMasterSubnet(&g_ble_mesh_access0_ctrl, friend_subnet_handle,
&master_subnet_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_master_subnet_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_DeleteNetKey()

fsp_err_t RM_BLE_MESH_ACCESS_DeleteNetKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle* )

To delete NetKey. This routine deletes a NetKey entry. Each NetKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::deleteNetKey.

Example:

```
/* Delete NetKey. */
   err = RM_BLE_MESH_ACCESS_DeleteNetKey(&g_ble_mesh_access0_ctrl, subnet_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetNetKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetNetKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t *const *p_net_key* )

To get NetKey. This routine fetches a NetKey entry. Each NetKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::getNetKey.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_ACCESS_GetNetKey(&g_ble_mesh_access0_ctrl, subnet_handle,
&net_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_net_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetNetKeyIndexList()

fsp_err_t RM_BLE_MESH_ACCESS_GetNetKeyIndexList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint16_t *const *p_netkey_count*, uint16_t *const *p_netkey_index_list* )

To get list of all known NetKeys. This routine returns a list of known NetKey Indices.

Implements rm_ble_mesh_access_api_t::getNetKeyIndexList.

Example:

```
/* Get list of all known NetKeys. */
   err = RM_BLE_MESH_ACCESS_GetNetKeyIndexList(&g_ble_mesh_access0_ctrl,
&netkey_count, &netkey_index_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_netkey_count and p_netkey_index_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_LookUpNid()

fsp_err_t RM_BLE_MESH_ACCESS_LookUpNid ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *nid*, rm_ble_mesh_network_subnet_handle_t *const *p_subnet_handle*, rm_ble_mesh_access_associated_keys_t *const *p_key_set* )

To search for NID. This routine searches for matching NID in subnet table.

Implements rm_ble_mesh_access_api_t::lookUpNid.

Example:

```
/* Search for NID. */
   err = RM_BLE_MESH_ACCESS_LookUpNid(&g_ble_mesh_access0_ctrl, nid, &subnet_handle,
&key_set);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_subnet_handle and p_key_set are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_LookUpNetworkId()

fsp_err_t RM_BLE_MESH_ACCESS_LookUpNetworkId ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t const *const *p_network_id*, rm_ble_mesh_network_subnet_handle_t *const *p_subnet_handle*, rm_ble_mesh_access_associated_keys_t *const *p_key_set* )

To search for Network ID. This routine searches for matching Network ID in subnet table.

Implements rm_ble_mesh_access_api_t::lookUpNetworkId.

Example:

```
/* Search for NID. */
   err = RM_BLE_MESH_ACCESS_LookUpNid(&g_ble_mesh_access0_ctrl, nid, &subnet_handle,
&key_set);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_network_id, p_subnet_handle and p_key_set are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_LookUpAid()

fsp_err_t RM_BLE_MESH_ACCESS_LookUpAid ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *aid*, rm_ble_mesh_network_appkey_handle_t *const *p_appkey_handle*, uint8_t *const *p_app_key* )

To search for AID. This routine searches for matching NID in subnet table.

Implements rm_ble_mesh_access_api_t::lookUpAid.

Example:

```
/* Search for AID. */
   err = RM_BLE_MESH_ACCESS_LookUpAid(&g_ble_mesh_access0_ctrl, aid, &appkey_handle,
&app_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_appkey_handle and p_app_key are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_SetProvisioningData()

fsp_err_t RM_BLE_MESH_ACCESS_SetProvisioningData ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_data_t const *const *p_prov_data* )

Set Provisioning Data. This routine configures the provisioning data with Access Layer.

Implements rm_ble_mesh_access_api_t::setProvisioningData.

Example:

```
/* Set provisioning data. */
   err = RM_BLE_MESH_ACCESS_SetProvisioningData(&g_ble_mesh_access0_ctrl,
&prov_data);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_prov_data is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetNid()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetNid ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_nid* )

To get NID associated with a subnet. This routine fetches the NID associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetNid.

Example:

```
/* Get NID associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetNid(&g_ble_mesh_access0_ctrl, subnet_handle,
&nid);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_nid is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetPrivacyKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetPrivacyKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_privacy_key* )

To get Privacy Key associated with a subnet. This routine fetches the Privacy Key associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetPrivacyKey.

Example:

```
/* Get Privacy Key associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetPrivacyKey(&g_ble_mesh_access0_ctrl,
subnet_handle, &privacy_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_privacy_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetNetworkId()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetNetworkId ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_network_id* )

To get Network ID associated with a subnet. This routine fetches the Network ID associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetNetworkId.

Example:

```
/* Get Network ID associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetNetworkId(&g_ble_mesh_access0_ctrl,
subnet_handle, &network_id);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_network_id is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetBeaconKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetBeaconKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_beacon_key* )

To get Beacon Key associated with a subnet. This routine fetches the Beacon Key associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetBeaconKey.

Example:

```
/* Get Beacon Key associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetBeaconKey(&g_ble_mesh_access0_ctrl,
subnet_handle, &beacon_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_beacon_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetIdentityKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetIdentityKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_identity_key* )

To get Identity Key associated with a subnet. This routine fetches the Identity Key associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetIdentityKey.

Example:

```
/* Get Identity Key associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetIdentityKey(&g_ble_mesh_access0_ctrl,
subnet_handle, &identity_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_identity_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetSubnetEncryptionKey()

fsp_err_t RM_BLE_MESH_ACCESS_GetSubnetEncryptionKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *handle*, uint8_t *const *p_encrypt_key* )

To get Encryption Key associated with a subnet. This routine fetches the Encryption Key associated with a subnet.

Implements rm_ble_mesh_access_api_t::getSubnetEncryptionKey.

Example:

```
/* Get Encryption Key associated with a subnet. */
   err = RM_BLE_MESH_ACCESS_GetSubnetEncryptionKey(&g_ble_mesh_access0_ctrl,
subnet_handle, &encrypt_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_encrypt_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetNodeIdentity()

fsp_err_t RM_BLE_MESH_ACCESS_GetNodeIdentity ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t *const *p_id_state* )

To get Node Identity. This routine gets Node Identity State of a node

Implements rm_ble_mesh_access_api_t::getNodeIdentity.

Example:

```
/* Get node identity. */
   err = RM_BLE_MESH_ACCESS_GetNodeIdentity(&g_ble_mesh_access0_ctrl, subnet_handle,
&id_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_id_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_SetNodeIdentity()

fsp_err_t RM_BLE_MESH_ACCESS_SetNodeIdentity ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t *const *p_id_state* )

To set Node Identity. This routine sets Node Identity State of a node.

Implements rm_ble_mesh_access_api_t::setNodeIdentity.

Example:

```
/* Set node identity. */
   err = RM_BLE_MESH_ACCESS_SetNodeIdentity(&g_ble_mesh_access0_ctrl, subnet_handle,
&id_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_id_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetKeyRefreshPhase()

fsp_err_t RM_BLE_MESH_ACCESS_GetKeyRefreshPhase ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t *const *p_key_refresh_state* )

To get Key Refresh Phase. This routine gets Key Refresh Phase State of a node

Implements rm_ble_mesh_access_api_t::getKeyRefreshPhase.

Example:

```
/* Get key refresh phase. */
   err = RM_BLE_MESH_ACCESS_GetKeyRefreshPhase(&g_ble_mesh_access0_ctrl,
subnet_handle, &key_refresh_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_key_refresh_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_SetKeyRefreshPhase()

fsp_err_t RM_BLE_MESH_ACCESS_SetKeyRefreshPhase ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t const *const *p_key_refresh_state* )

To set Key Refresh Phase. This routine sets Key Refresh Phase State of a node.

Implements rm_ble_mesh_access_api_t::setKeyRefreshPhase.

Example:

```
/* Set key refresh phase. */
   err = RM_BLE_MESH_ACCESS_SetKeyRefreshPhase(&g_ble_mesh_access0_ctrl,
subnet_handle, &key_refresh_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_key_refresh_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_SetTransmitState()

fsp_err_t RM_BLE_MESH_ACCESS_SetTransmitState ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *tx_state_type*, uint8_t *tx_state* )

To set Network/Relay Transmit state. This routine sets Network/Relay Transmit state.

Implements rm_ble_mesh_access_api_t::setTransmitState.

Example:

```
/* Set network/relay transmit state. */
   err = RM_BLE_MESH_ACCESS_SetTransmitState(&g_ble_mesh_access0_ctrl,
tx_state_type, tx_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_GetTransmitState()

fsp_err_t RM_BLE_MESH_ACCESS_GetTransmitState ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint8_t *tx_state_type*, uint8_t *const *p_tx_state* )

To get Network/Relay Transmit state. This routine gets Network/Relay Transmit state.

Implements rm_ble_mesh_access_api_t::getTransmitState.

Example:

```
/* Get network/relay transmit state. */
   err = RM_BLE_MESH_ACCESS_GetTransmitState(&g_ble_mesh_access0_ctrl,
tx_state_type, &tx_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_tx_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_AddAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_AddAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *appkey_index*, uint8_t const *const *p_app_key* )

To add AppKey. This routine adds AppKey entry. Each AppKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::addAppKey.

Example:

```
/* Add AppKey. */
   err = RM_BLE_MESH_ACCESS_AddAppKey(&g_ble_mesh_access0_ctrl, subnet_handle,
appkey_index, &app_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_app_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_UpdateAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_UpdateAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *appkey_index*, uint8_t const *const *p_app_key* )

To update AppKey. This routine updates AppKey entry. Each AppKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::updateAppKey.

Example:

```
/* Update AppKey. */
   err = RM_BLE_MESH_ACCESS_UpdateAppKey(&g_ble_mesh_access0_ctrl, subnet_handle,
appkey_index, &app_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_app_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_DeleteAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_DeleteAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *appkey_index*, uint8_t const *const *p_app_key* )

To delete AppKey. This routine deletes AppKey entry. Each AppKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::deleteAppKey.

Example:

```
/* Delete AppKey. */
   err = RM_BLE_MESH_ACCESS_DeleteAppKey(&g_ble_mesh_access0_ctrl, subnet_handle,
appkey_index, &app_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_app_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetAppKeyHandle()

fsp_err_t RM_BLE_MESH_ACCESS_GetAppKeyHandle ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *appkey_index*, uint8_t const *const *p_app_key*, rm_ble_mesh_network_appkey_handle_t *const *p_appkey_handle* )

To get AppKey Handle for a given AppKey Index. This routine gets AppKey Handle for a given AppKey Index. Each AppKey is associated with a subnet.

Implements rm_ble_mesh_access_api_t::getAppKeyHandle.

Example:

```
/* Get AppKey handle for a given AppKey index. */
   err = RM_BLE_MESH_ACCESS_GetAppKeyHandle(&g_ble_mesh_access0_ctrl,
                                            subnet_handle,
                                            appkey_index,
                                            &app_key,
                                            &appkey_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_appkey_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetAppKeyIndexList()

fsp_err_t RM_BLE_MESH_ACCESS_GetAppKeyIndexList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint16_t *const *p_appkey_count*, uint16_t *const *p_appkey_index_list* )

To get list of all known AppKeys. This routine returns a list of known AppKey Indices associated with a subnet.

Implements rm_ble_mesh_access_api_t::getAppKeyIndexList.

Example:

```
/* Get list of all known AppKeys. */
   err = RM_BLE_MESH_ACCESS_GetAppKeyIndexList(&g_ble_mesh_access0_ctrl,
                                               subnet_handle,
                                                &appkey_count,
                                                &appkey_index_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_appkey_count and p_appkey_index_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_BindModelWithAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_BindModelWithAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, uint16_t *appkey_index* )

To bind a model with an AppKey. This routine binds a model with an AppKey.

Implements rm_ble_mesh_access_api_t::bindModelWithAppKey.

Example:

```
/* Bind a model with an AppKey. */
   err = RM_BLE_MESH_ACCESS_BindModelWithAppKey(&g_ble_mesh_access0_ctrl,
model_handle, appkey_index);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_UnbindModelWithAppKey()

fsp_err_t RM_BLE_MESH_ACCESS_UnbindModelWithAppKey ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, uint16_t *appkey_index* )

To unbind a model with an AppKey. This routine unbinds a model with an AppKey.

Implements rm_ble_mesh_access_api_t::unbindModelWithAppKey.

Example:

```
/* Unbind a model with an AppKey. */
   err = RM_BLE_MESH_ACCESS_UnbindModelWithAppKey(&g_ble_mesh_access0_ctrl,
model_handle, appkey_index);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetModelAppKeyList()

fsp_err_t RM_BLE_MESH_ACCESS_GetModelAppKeyList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, uint16_t *const *p_appkey_count*, uint16_t *const *p_appkey_index_list* )

To get list of all AppKeys associated with a model. This routine returns a list of known AppKey Indices associated with a model.

Implements rm_ble_mesh_access_api_t::getModelAppKeyList.

Example:

```
/* Get list of all AppKeys associated with a model. */
   err = RM_BLE_MESH_ACCESS_GetModelAppKeyList(&g_ble_mesh_access0_ctrl,
                                               model_handle,
                                               &appkey_count,
                                               &appkey_index_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_appkey_count and p_appkey_index_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_SetModelPublication()

fsp_err_t RM_BLE_MESH_ACCESS_SetModelPublication ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, rm_ble_mesh_access_publish_info_t *const *p_publish_info* )

To set Publication information associated with a model. This routine sets Publication information associated with a model.

Implements rm_ble_mesh_access_api_t::setModelPublication.

Example:

```
/* Set publication information associated with a model. */
   err = RM_BLE_MESH_ACCESS_SetModelPublication(&g_ble_mesh_access0_ctrl,
model_handle, &publish_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_publish_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_SetModelPublicationPeriodDivisor()

fsp_err_t RM_BLE_MESH_ACCESS_SetModelPublicationPeriodDivisor ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, uint8_t *period_divisor* )

To set Publication Fast Period Divisor information associated with a model. This routine sets Publication Fast Period Divisor information associated with a model.

Implements rm_ble_mesh_access_api_t::setModelPublicationPeriodDivisor.

Example:

```
/* Set publication fast period divisor information associated with a model. */
   err =
RM_BLE_MESH_ACCESS_SetModelPublicationPeriodDivisor(&g_ble_mesh_access0_ctrl,
model_handle, period_divisor);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_GetModelPublication()

fsp_err_t RM_BLE_MESH_ACCESS_GetModelPublication ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, rm_ble_mesh_access_publish_info_t *const *p_publish_info* )

To get Publication information associated with a model. This routine returns Publication information associated with a model.

Implements rm_ble_mesh_access_api_t::getModelPublication.

Example:

```
/* Get publication information associated with a model. */
   err = RM_BLE_MESH_ACCESS_GetModelPublication(&g_ble_mesh_access0_ctrl,
model_handle, &publish_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_publish_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_AddModelSubscription()

fsp_err_t RM_BLE_MESH_ACCESS_AddModelSubscription ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, rm_ble_mesh_access_address_t const *const *p_sub_addr* )

To add an address to a model subscription list. This routine adds an address to a subscription list of a model.

Implements rm_ble_mesh_access_api_t::addModelSubscription.

Example:

```
/* Add an address to a model subscription list. */
   err = RM_BLE_MESH_ACCESS_AddModelSubscription(&g_ble_mesh_access0_ctrl,
model_handle, &sub_addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_sub_addr is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_DeleteModelSubscription()

fsp_err_t RM_BLE_MESH_ACCESS_DeleteModelSubscription ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, rm_ble_mesh_access_address_t const *const *p_sub_addr* )

To delete an address to a model subscription list. This routine deletes an address to a subscription list of a model.

Implements rm_ble_mesh_access_api_t::deleteModelSubscription.

Example:

```
/* Delete an address to a model subscription list. */
   err = RM_BLE_MESH_ACCESS_DeleteModelSubscription(&g_ble_mesh_access0_ctrl,
model_handle, &sub_addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_sub_addr is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_DeleteAllModelSubscription()

fsp_err_t RM_BLE_MESH_ACCESS_DeleteAllModelSubscription ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle* )

To discard a model subscription list. This routine discards a subscription list of a model.

Implements rm_ble_mesh_access_api_t::deleteAllModelSubscription.

Example:

```
/* Discard a model subscription list. */
   err = RM_BLE_MESH_ACCESS_DeleteAllModelSubscription(&g_ble_mesh_access0_ctrl,
model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetModelSubscriptionList()

fsp_err_t RM_BLE_MESH_ACCESS_GetModelSubscriptionList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *model_handle*, uint16_t *const *p_sub_addr_count*, uint16_t *const *p_sub_addr_list* )

To get list of subscription addresses of a model. This routine returns a list of subscription addresses of a model.

Implements rm_ble_mesh_access_api_t::getModelSubscriptionList.

Example:

```
/* Get list of subscription addresses of a model. */
   err = RM_BLE_MESH_ACCESS_GetModelSubscriptionList(&g_ble_mesh_access0_ctrl,
                                                     model_handle,
                                                         &sub_addr_count,
                                                         &sub_addr_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_sub_addr_count and p_sub_addr_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_GetAllModelSubscriptionList()

fsp_err_t RM_BLE_MESH_ACCESS_GetAllModelSubscriptionList ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, uint16_t *const *p_sub_addr_count*, uint16_t *const *p_sub_addr_list* )

To get list of subscription addresses of all the models. This routine returns a consolidated list of subscription addresses of all the models.

Implements rm_ble_mesh_access_api_t::getAllModelSubscriptionList.

Example:

```
/* Get list of subscription addresses of all the models. */
   err = RM_BLE_MESH_ACCESS_GetAllModelSubscriptionList(&g_ble_mesh_access0_ctrl,
&sub_addr_count, &sub_addr_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_sub_addr_count and p_sub_addr_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_ACCESS_IsValidElementAddress()

fsp_err_t RM_BLE_MESH_ACCESS_IsValidElementAddress ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr* )

To check if valid element address to receive a packet. This routine checks if destination address in a received packet matches with any of the known element address of local or friend device.

Implements rm_ble_mesh_access_api_t::isValidElementAddress.

Example:

```
/* Check if valid element address to receive a packet. */
   err = RM_BLE_MESH_ACCESS_IsValidElementAddress(&g_ble_mesh_access0_ctrl, addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |

### ◆ RM_BLE_MESH_ACCESS_IsFixedGroupAddressToBeProcessed()

fsp_err_t RM_BLE_MESH_ACCESS_IsFixedGroupAddressToBeProcessed ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr* )

To check if Fixed Group Address in receive packet to be processed. This routine checks if destination address in a received packet as a Fixed Group Address to be processed.

Implements rm_ble_mesh_access_api_t::isFixedGroupAddressToBeProcessed.

Example:

```
/* Check if fixed group address in receive packet to be processed. */
   err =
RM_BLE_MESH_ACCESS_IsFixedGroupAddressToBeProcessed(&g_ble_mesh_access0_ctrl, addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_ACCESS_IsValidSubscriptionAddress()

fsp_err_t RM_BLE_MESH_ACCESS_IsValidSubscriptionAddress ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr* )

To check if valid subscription address to receive a packet. This routine checks if destination address in a received packet matches with any of the known subscription address of local or friend device.

Implements rm_ble_mesh_access_api_t::isValidSubscriptionAddress.

Example:

```
/* Check if valid subscription address to receive a packet. */
   err = RM_BLE_MESH_ACCESS_IsValidSubscriptionAddress(&g_ble_mesh_access0_ctrl,
addr);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_ACCESS_EnableIvUpdateTestMode()

fsp_err_t RM_BLE_MESH_ACCESS_EnableIvUpdateTestMode ( rm_ble_mesh_access_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_iv_update_test_mode_t *mode* )

To set the IV Update Test Mode feature. This routine is used to set the IV Update Test Mode flag.

Implements rm_ble_mesh_access_api_t::enableIvUpdateTestMode.

Example:

```
/* Set the IV update test mode feature. */
   err = RM_BLE_MESH_ACCESS_EnableIvUpdateTestMode(&g_ble_mesh_access0_ctrl, mode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Bearer (rm_ble_mesh_bearer)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_BEARER_Open (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_BEARER_Close (rm_ble_mesh_bearer_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_BEARER_RegisterInterface (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_ntf_callback_result_t (*p_callback)(rm_ble_mesh_bearer_ntf_callback_args_t *p_args)) |
| fsp_err_t | RM_BLE_MESH_BEARER_RegisterBeaconHandler (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_beacon_type_t bcon_type, void(*p_handler)(rm_ble_mesh_bearer_beacon_callback_args_t *p_args)) |
| fsp_err_t | RM_BLE_MESH_BEARER_AddBearer (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_info_t const *const p_brr_info, rm_ble_mesh_bearer_handle_t *const p_brr_handle) |
| fsp_err_t | RM_BLE_MESH_BEARER_RemoveBearer (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_handle_t const *const p_brr_handle) |
| fsp_err_t | RM_BLE_MESH_BEARER_ObserveBeacon (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t bcon_type, uint8_t enable) |
| fsp_err_t | RM_BLE_MESH_BEARER_BcastUnprovisionedBeacon (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_dev_uuid, uint16_t oob_info, rm_ble_mesh_buffer_t const *const p_uri) |
| fsp_err_t | RM_BLE_MESH_BEARER_BroadcastBeacon (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_packet, uint16_t length) |
| fsp_err_t | RM_BLE_MESH_BEARER_StartProxyAdv (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_data, uint16_t datalen) |
| fsp_err_t | RM_BLE_MESH_BEARER_SendPdu (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_handle_t const *const p_brr_handle, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_buffer_t const *const p_buffer) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_BEARER_GetPacketRssi (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t *p_rssi_value) |
| fsp_err_t | RM_BLE_MESH_BEARER_Sleep (rm_ble_mesh_bearer_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_BEARER_Wakeup (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t mode) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Bearer module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_bearer

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_bearer_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Bearer (rm_ble_mesh_bearer)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Bearer (rm_ble_mesh_bearer).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_bearer0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_bearer_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_ble_mesh_bearer_instance_ctrl_t

| struct rm_ble_mesh_bearer_instance_ctrl_t |
|---|
| RM_BLE_MESH_BEARER private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_BEARER_Open() is called. |

### Function Documentation

#### ◆ RM_BLE_MESH_BEARER_Open()

fsp_err_t RM_BLE_MESH_BEARER_Open ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_cfg_t const *const *p_cfg* )

Open bearer middleware.

Implements rm_ble_mesh_bearer_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_BEARER_Open(&g_ble_mesh_bearer0_ctrl, &g_ble_mesh_bearer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |

#### ◆ RM_BLE_MESH_BEARER_Close()

fsp_err_t RM_BLE_MESH_BEARER_Close ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*)

Close bearer middleware. Implements rm_ble_mesh_bearer_api_t::close.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_BEARER_Close(&g_ble_mesh_bearer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_BEARER_RegisterInterface()

fsp_err_t RM_BLE_MESH_BEARER_RegisterInterface ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_type_t *brr_type*, rm_ble_mesh_bearer_ntf_callback_result_t(*)( rm_ble_mesh_bearer_ntf_callback_args_t *p_args) *p_callback* )

Register interface with Bearer Layer. This routine registers interface with the Bearer Layer. Bearer Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_bearer_api_t::registerInterface.

Example:

```
/* Register Bearer interface. */
   err = RM_BLE_MESH_BEARER_RegisterInterface(&g_ble_mesh_bearer0_ctrl, brr_type,
rm_ble_mesh_bearer_ntf_callback);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_BEARER_RegisterBeaconHandler()

fsp_err_t RM_BLE_MESH_BEARER_RegisterBeaconHandler ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_beacon_type_t *bcon_type*, void(*)(rm_ble_mesh_bearer_beacon_callback_args_t *p_args) *p_handler* )

Register beacon interface with Bearer Layer. This routine registers interface with the Bearer Layer to process Beacons. Bearer Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_bearer_api_t::registerBeaconHandler.

Example:

```
/* Register beacon handler. */
   err = RM_BLE_MESH_BEARER_RegisterBeaconHandler(&g_ble_mesh_bearer0_ctrl,
RM_BLE_MESH_BEARER_TYPE_BCON,
                                                  rm_ble_mesh_bearer_beacon_callback
);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_AddBearer()

fsp_err_t RM_BLE_MESH_BEARER_AddBearer ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_type_t *brr_type*, rm_ble_mesh_bearer_info_t const *const *p_brr_info*, rm_ble_mesh_bearer_handle_t *const *p_brr_handle* )

Add a bearer to Bearer Layer. This routine adds a bearer that is setup by the application for use by the Mesh Stack. Bearer Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_bearer_api_t::addBearer.

Example:

```
/* Add Bearer. */
   err = RM_BLE_MESH_BEARER_AddBearer(&g_ble_mesh_bearer0_ctrl,
RM_BLE_MESH_BEARER_TYPE_BCON, &brr_info, &brr_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_brr_info and p_brr_handle are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_RemoveBearer()

fsp_err_t RM_BLE_MESH_BEARER_RemoveBearer ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_type_t *brr_type*, rm_ble_mesh_bearer_handle_t const *const *p_brr_handle* )

Remove a bearer from Bearer Layer. This routine removes a bearer from the Mesh Stack. Bearer Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_bearer_api_t::removeBearer.

Example:

```
/* Remove Bearer. */
   err = RM_BLE_MESH_BEARER_RemoveBearer(&g_ble_mesh_bearer0_ctrl,
RM_BLE_MESH_BEARER_TYPE_BCON, &brr_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_brr_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_ObserveBeacon()

fsp_err_t RM_BLE_MESH_BEARER_ObserveBeacon ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t *bcon_type*, uint8_t *enable* )

Observe on/off for the beacon type. This routine sends enables/disables the observation procedure for the given beacon type.

Implements rm_ble_mesh_bearer_api_t::observeBeacon.

Example:

```
/* Observe beacon. */
   err = RM_BLE_MESH_BEARER_ObserveBeacon(&g_ble_mesh_bearer0_ctrl, bcon_type,
enable);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_BcastUnprovisionedBeacon()

fsp_err_t RM_BLE_MESH_BEARER_BcastUnprovisionedBeacon ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t *type*, uint8_t const *const *p_dev_uuid*, uint16_t *oob_info*, rm_ble_mesh_buffer_t const *const *p_uri* )

API to send unprovisioned device beacon. This routine sends Unprovisioned Device Beacon.

Implements rm_ble_mesh_bearer_api_t::bcastUnprovisionedBeacon.

Example:

```
/* Broadcast unprovisioned beacon. */
   err = RM_BLE_MESH_BEARER_BcastUnprovisionedBeacon(&g_ble_mesh_bearer0_ctrl, type,
&dev_uuid, oob_info, &uri);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_dev_uuid and p_uri are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_BroadcastBeacon()

fsp_err_t RM_BLE_MESH_BEARER_BroadcastBeacon ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t *type*, uint8_t const *const *p_packet*, uint16_t *length* )

API to broadcast a beacon. This routine sends the beacon of given type on Adv and GATT bearers.

Implements rm_ble_mesh_bearer_api_t::broadcastBeacon.

Example:

```
/* Broadcast beacon. */
   err = RM_BLE_MESH_BEARER_BroadcastBeacon(&g_ble_mesh_bearer0_ctrl, type, &packet,
length);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_packet is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_BEARER_StartProxyAdv()

fsp_err_t RM_BLE_MESH_BEARER_StartProxyAdv ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t *type*, uint8_t const *const *p_data*, uint16_t *datalen* )

API to send proxy device ADV. This routine sends Proxy Device ADV.

Implements rm_ble_mesh_bearer_api_t::startProxyAdv.

Example:

```
/* Start proxy advertising. */
   err = RM_BLE_MESH_BEARER_StartProxyAdv(&g_ble_mesh_bearer0_ctrl, type, &data,
length);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_data is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_BEARER_SendPdu()

fsp_err_t RM_BLE_MESH_BEARER_SendPdu ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, rm_ble_mesh_bearer_handle_t const *const *p_brr_handle*, rm_ble_mesh_bearer_type_t *brr_type*, rm_ble_mesh_buffer_t const *const *p_buffer* )

Send a bearer PDU. This routine sends a PDU from the Mesh stack to over the bearer indicated by the bearer handle.

Implements rm_ble_mesh_bearer_api_t::sendPdu.

Example:

```
/* Send common Bearer PDUs. */
   err = RM_BLE_MESH_BEARER_SendPdu(&g_ble_mesh_bearer0_ctrl, &brr_handle,
RM_BLE_MESH_BEARER_TYPE_BCON, &buffer);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_brr_handle and p_buffer are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_BEARER_GetPacketRssi()

fsp_err_t RM_BLE_MESH_BEARER_GetPacketRssi ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t * *p_rssi_value* )

Get the RSSI of current received packet being processed. This routine returns the RSSI value of the received packet in its context when called from the Mesh stack.

Implements rm_ble_mesh_bearer_api_t::getPacketRssi.

Example:

```
/* Get the RSSI of current received packet being processed. */
   err = RM_BLE_MESH_BEARER_GetPacketRssi(&g_ble_mesh_bearer0_ctrl, &rssi_value);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_rssi_value is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_Sleep()

fsp_err_t RM_BLE_MESH_BEARER_Sleep ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*)

Put the bearer to sleep. This routine requests the underlying bearer interface to sleep. Default bearer interface is that of advertising bearer.

Implements rm_ble_mesh_bearer_api_t::sleep.

Example:

```
/* Put the bearer to sleep. */
   err = RM_BLE_MESH_BEARER_Sleep(&g_ble_mesh_bearer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_BEARER_Wakeup()

fsp_err_t RM_BLE_MESH_BEARER_Wakeup ( rm_ble_mesh_bearer_ctrl_t *const *p_ctrl*, uint8_t *mode* )

Wakeup the bearer. This routine requests the underlying bearer interface to wakeup. Default bearer interface is that of advertising bearer.

Implements rm_ble_mesh_bearer_api_t::wakeup.

Example:

```
/* Wakeup the bearer. */
   err = RM_BLE_MESH_BEARER_Wakeup(&g_ble_mesh_bearer0_ctrl, mode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Bearer Platform (rm_mesh_bearer_platform)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_BEARER_PLATFORM_Open (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_Close (rm_mesh_bearer_platform_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_Setup (rm_mesh_bearer_platform_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_CallbackSet (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_gatt_iface_cb_t callback) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_SetGattMode (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_gatt_mode_t mode) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_GetGattMode (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_gatt_mode_t *p_mode) |

| | |
|---|---|
| fsp_err_t | RM_MESH_BEARER_PLATFORM_SetScanResponseData (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_data, uint8_t len) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_ScanGattBearer (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_state_t state, rm_mesh_bearer_platform_gatt_mode_t mode) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_Connect (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_remote_address, uint8_t address_type, rm_mesh_bearer_platform_gatt_mode_t mode) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_DiscoverService (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_gatt_mode_t mode) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_ConfigureNotification (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_state_t state, rm_mesh_bearer_platform_gatt_mode_t mode) |
| fsp_err_t | RM_MESH_BEARER_PLATFORM_Disconnect (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Bearer Platform module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_bearer_platform

The following build time configurations are defined in fsp_cfg/rm_mesh_bearer_platform_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Debug Public Address | Must be a valid device address | FF:FF:FF:50:90:74 | Public Address of firmware initial value. |
| Debug Random Address | Must be a valid device address | FF:FF:FF:FF:FF:FF | Random Address of firmware initial value. |
| Maximum number of | Value must be an | 7 | Maximum number of |

| | | | |
|---|---|---|---|
| connections | integer between 1 and 7, and lower than the value defined in ble module. | | connections. |
| Maximum connection data length | Value must be an integer between 27 and 251, and lower than the value defined in ble module. | 251 | Maximum connection data length. |
| Maximum advertising data length | Value must be an integer between 31 and 1650, and lower than the value defined in ble module. | 1650 | Maximum advertising data length. |
| Maximum advertising set number | Value must be an integer between 1 and 4, and lower than the value defined in ble module. | 4 | Maximum advertising set number. |
| Maximum periodic sync set number. | Value must be an integer between 1 and 2, and lower than the value defined in ble module. | 2 | Maximum periodic sync set number. |
| Store Security Data | • Disable<br>• Enable | Disable | Store Security Data in DataFlash. |
| Data Flash Block for Security Data | Value must be an integer between 0 and 7, and lower than the value defined in ble module. | 0 | Data Flash Block for Security Data Management. |
| Remote Device Bonding Number | Value must be an integer between 1 and 7, and lower than the value defined in ble module. | 7 | Number of remote device bonding information. |
| Connection Event Start Notify | • Disable<br>• Enable | Disable | Set Connection event start notify enable/disable. |
| Connection Event Close Notify | • Disable<br>• Enable | Disable | Set Connection event close notify enable/disable. |
| Advertising Event Start Notify | • Disable<br>• Enable | Disable | Set Advertising event start notify enable/disable. |
| Advertising Event Close Notify | • Disable<br>• Enable | Disable | Set Advertising event close notify enable/disable. |

| | | | |
|---|---|---|---|
| Scanning Event Start Notify | • Disable<br>• Enable | Disable | Set Scanning event start notify enable/disable. |
| Scanning Event Close Notify | • Disable<br>• Enable | Disable | Set Scanning event close notify enable/disable. |
| Initiating Event Start Notify | • Disable<br>• Enable | Disable | Set Initiating event start notify enable/disable. |
| Initiating Event Close Notify | • Disable<br>• Enable | Disable | Set Initiating event close notify enable/disable. |
| RF Deep Sleep Start Notify | • Disable<br>• Enable | Disable | Set RF_DEEP_SLEEP start notify enable/disable. |
| RF Deep Sleep Wakeup Notify | • Disable<br>• Enable | Disable | Set RF_DEEP_SLEEP wakeup notify enable/disable. |
| Bluetooth dedicated clock | Value must be an integer between 0 and 15, and lower than the value defined in ble module. | 6 | Load capacitance adjustment. |
| DC-DC converter | • Disable<br>• Enable | Disable | Set DC-DC converter for RF part. |
| Slow Clock Source | • Use RF_LOCO<br>• Use External 32.768kHz | Use RF_LOCO | Set slow clock source for RF part. |
| MCU CLKOUT Port | • P109<br>• P205 | P109 | When MESH_BEARER_PLATFORM_CFG_RF_EXTERNAL_32K_ENABLE = 1, Set port of MCU CLKOUT. |
| MCU CLKOUT Frequency Output | • MCU CLKOUT frequency 32.768kHz<br>• MCU CLKOUT frequency 16.384kHz | MCU CLKOUT frequency 32.768kHz | When MESH_BEARER_PLATFORM_CFG_RF_EXTERNAL_32K_ENABLE = 1, set frequency output from CLKOUT of MCU part. |
| Sleep Clock Accuracy(SCA) | Value must be an integer between 0 and 500, and lower than the value defined in ble module. | 250 | When MESH_BEARER_PLATFORM_CFG_RF_EXTERNAL_32K_ENABLE = 1, set Sleep Clock Accuracy(SCA) for RF slow clock. |
| Transmission Power Maximum Value | • max +0dBm<br>• max +4dBm | max +4dBm | Set transmission power maximum value. |

| | | | |
|---|---|---|---|
| Transmission Power Default Value | • High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm)<br>• Mid 0dBm(Transmission Power Maximum Value = +0dBm) / 0dBm(Transmission Power Maximum Value = +4dBm)<br>• Low -18dBm(Transmission Power Maximum Value = +0dBm) / -20dBm(Transmission Power Maximum Value = +4dBm) | High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm) | Set default transmit power. Default transmit power is dependent on the configuration of Maximum transmission power(MESH_BEARER_PLATFORM_CFG_RF_MAX_TX_POW). |
| CLKOUT_RF Output | • No output<br>• 4MHz output<br>• 2MHz output<br>• 1MHz output | No output | Set CLKOUT_RF output setting. |
| RF_DEEP_SLEEP Transition | • Disable<br>• Enable | Enable | Set RF_DEEP_SLEEP transition. |
| MCU Main Clock Frequency | Value must be an integer between 1000 and 20000, and lower than the value defined in ble module. | 8000 | Set MCU Main Clock Frequency (kHz). Set clock source according to your board environment. HOCO: don't care. / Main Clock: 1000 to 20000 kHz / PLL Circuit: 4000 to 12500 kHz |
| Code Flash(ROM) Device Data Block | Value must be an integer between -1 and 255, and lower than the value defined in ble module. | 255 | Device specific data block on Code Flash (ROM). |
| Device Specific Data Flash Block | Value must be an integer between -1 and 7, and lower than the value defined in ble module. | -1 | Device specific data block on E2 Data Flash. |

| | | | |
|---|---|---|---|
| MTU Size Configured | Value must be an integer between 23 and 247, and lower than the value defined in ble module. | 247 | MTU Size configured by GATT MTU exchange procedure. |
| Timer Slot Maximum Number | Value must be an integer between 1 and 10, and lower than the value defined in ble module. | 10 | The maximum number of timer slot. |

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Bearer Platform (rm_mesh_bearer_platform)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Bearer Platform (rm_mesh_bearer_platform).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_bearer_platform0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Device Address Type | Invalid Device Address Type | 0 | Select device address type ( public : 0, random : 1). |
| GATT Server Callback Number | Invalid Callback Number | 1 | Number of GATT server callback. |
| GATT Client Callback Number | Invalid Callback Number | 1 | Number of GATT client callback. |
| Vender Specific Callback | Name Must Be a Valid C Symbol. | NULL | Vendor specific callback function name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_bearer_platform_instance_ctrl_t |
| struct | rm_mesh_bearer_platform_extended_cfg_t |

**Typedefs**

| | |
|---|---|
| typedef void(* | rm_mesh_bearer_platform_gatt_iface_cb_t) (uint8_t event_name, uint8_t event_param, uint16_t conn_hdl, st_ble_dev_addr_t *peer_addr)<br>Callback function for GATT interface event. More... |

## Enumerations

| | |
|---|---|
| enum | rm_mesh_bearer_platform_interface_event_t |

## Data Structure Documentation

### ◆ rm_mesh_bearer_platform_instance_ctrl_t

struct rm_mesh_bearer_platform_instance_ctrl_t

RM_BLE_MESH_BEARER private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_BEARER_Open() is called.

### ◆ rm_mesh_bearer_platform_extended_cfg_t

struct rm_mesh_bearer_platform_extended_cfg_t

Bearer port extension for renesas BLE stack.

## Typedef Documentation

### ◆ rm_mesh_bearer_platform_gatt_iface_cb_t

typedef void(* rm_mesh_bearer_platform_gatt_iface_cb_t) (uint8_t event_name, uint8_t event_param, uint16_t conn_hdl, st_ble_dev_addr_t *peer_addr)

Callback function for GATT interface event.

**Parameters**

| | |
|---|---|
| event_name | The event defined by rm_mesh_bearer_platform_interface_event_t |
| event_param | The mode of GATT connection rm_mesh_bearer_platform_gatt_mode_t |
| conn_hdl | The connection handle |
| peer_addr | Pointer to the connected device address |

## Enumeration Type Documentation

### ◆ rm_mesh_bearer_platform_interface_event_t

| enum rm_mesh_bearer_platform_interface_event_t | |
|---|---|
| GATT Interface Events | |
| Enumerator | |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_UP | The event GATT Bearer BLE Link Layer connected |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_DOWN | The event GATT Bearer BLE Link Layer disconnected |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_ENABLE | The event GATT Bearer service enabled for communication |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_DISABLE | The event GATT Bearer service disabled for communication |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_NOT_FOUND | The event discovery process is not completed |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_SCAN | The event that connectable device having Mesh GATT Service is found |
| RM_MESH_BEARER_PLATFORM_INTERFACE_EVENT_CANCEL | The Event GATT/BLE link layer connection creation is canceled |

## Function Documentation

### ◆ RM_MESH_BEARER_PLATFORM_Open()

fsp_err_t RM_MESH_BEARER_PLATFORM_Open ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, rm_mesh_bearer_platform_cfg_t const *const *p_cfg* )

Open Bearer Platform middleware. Initialize underlying BLE Protocol Stack to use as a Mesh Bearer. API to initialize underlying BLE Protocol Stack to use as a Mesh Bearer. Completion of the initialization is notified by the callback function.

Implements rm_mesh_bearer_platform_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_BEARER_PLATFORM_Open(&g_mesh_bearer_platform0_ctrl,
&g_mesh_bearer_platform0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_MESH_BEARER_PLATFORM_Close()

fsp_err_t RM_MESH_BEARER_PLATFORM_Close ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*)

Close Bearer Platform middleware. Terminate BLE stack.

Implements rm_mesh_bearer_platform_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_BEARER_PLATFORM_Close(&g_mesh_bearer_platform0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_Setup()

fsp_err_t RM_MESH_BEARER_PLATFORM_Setup ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*)

Register ADV bearer with Mesh stack and start scan. API to register ADV bearer with Mesh stack. After registering, this routine starts Scan.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_bearer_platform control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_CallbackSet()

fsp_err_t RM_MESH_BEARER_PLATFORM_CallbackSet ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, rm_mesh_bearer_platform_gatt_iface_cb_t *callback* )

Register callback function to receive GATT interface events. API to register callback function to receive GATT interface events.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_bearer_platform control block. |
| [in] | callback | Callback function. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_SetGattMode()

fsp_err_t RM_MESH_BEARER_PLATFORM_SetGattMode ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, rm_mesh_bearer_platform_gatt_mode_t *mode* )

Set GATT Bearer mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_bearer_platform control block. |
| [in] | mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_GetGattMode()

fsp_err_t RM_MESH_BEARER_PLATFORM_GetGattMode ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, rm_mesh_bearer_platform_gatt_mode_t * *p_mode* )

Get current GATT Bearer mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_bearer_platform control block. |
| [in] | p_mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_mode is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_SetScanResponseData()

fsp_err_t RM_MESH_BEARER_PLATFORM_SetScanResponseData ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, uint8_t * *p_data*, uint8_t *len* )

Set scan response data in connectable and scannable undirected advertising event. API to Set Scan Response Data in Connectable and scannable undirected Advertising event. Scan Response Data can be used for indicating additional information such as << Complete Local Name>>.

Implements rm_mesh_bearer_platform_api_t::setScanResponseData.

Example:

```
 /* Set scan response data in connectable and scannable undirected advertising
event.. */
    err = RM_MESH_BEARER_PLATFORM_SetScanResponseData(&g_mesh_bearer_platform0_ctrl,
&data, length);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_data is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_ScanGattBearer()

fsp_err_t RM_MESH_BEARER_PLATFORM_ScanGattBearer ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, rm_mesh_bearer_platform_state_t *state*, rm_mesh_bearer_platform_gatt_mode_t *mode* )

Manage reporting connectable device having Mesh GATT service. API to manage reporting connectable device having Mesh GATT service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_bearer_platform control block. |
| [in] | state | Notification configuration flag; enable if RM_MESH_BEARER_PLATFORM_STATE_ENABLE, or disable if RM_MESH_BEARER_PLATFORM_STATE_DISABLE. |
| [in] | mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_Connect()

fsp_err_t RM_MESH_BEARER_PLATFORM_Connect ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, uint8_t * *p_remote_address*, uint8_t *address_type*, rm_mesh_bearer_platform_gatt_mode_t *mode* )

Request to create connection. API to request to Create Connection. Completion of the establishing a connection is notified by RM_BLE_MESH_BEARER_IFACE_UP. After establishing a connection, RM_MESH_BEARER_PLATFORM_DiscoverService() is invoked automatically.

Implements rm_mesh_bearer_platform_api_t::connect.

Example:

```
/* Request to create connection. */
   err = RM_MESH_BEARER_PLATFORM_Connect(&g_mesh_bearer_platform0_ctrl,
                                         &remote_address,
                                         bearer_address_type,
RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_remote_address is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_DiscoverService()

fsp_err_t RM_MESH_BEARER_PLATFORM_DiscoverService ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, uint16_t *handle*, rm_mesh_bearer_platform_gatt_mode_t *mode* )

Start service discovery for Mesh GATT service. API to start Service Discovery for Mesh GATT Service. If Mesh GATT Service specified by the "mode" argument, this routine enables Notification of the Mesh GATT Service by invoking RM_MESH_BEARER_PLATFORM_ConfigureNotification().

Implements rm_mesh_bearer_platform_api_t::discoverService.

Example:

```
/* Start service discovery for Mesh GATT service. */
   err = RM_MESH_BEARER_PLATFORM_DiscoverService(&g_mesh_bearer_platform0_ctrl,
                                                 handle,
RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_ConfigureNotification()

fsp_err_t RM_MESH_BEARER_PLATFORM_ConfigureNotification ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, uint16_t *handle*, rm_mesh_bearer_platform_state_t *state*, rm_mesh_bearer_platform_gatt_mode_t *mode* )

Configure GATT notification of Mesh GATT service. API to configure GATT Notification of Mesh GATT Service.

Implements rm_mesh_bearer_platform_api_t::configureNotification.

Example:

```
/* Configure GATT notification of Mesh GATT service. */
   err =
RM_MESH_BEARER_PLATFORM_ConfigureNotification(&g_mesh_bearer_platform0_ctrl,
                                              handle,
 RM_MESH_BEARER_PLATFORM_STATE_ENABLE,
 RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_MESH_BEARER_PLATFORM_Disconnect()

fsp_err_t RM_MESH_BEARER_PLATFORM_Disconnect ( rm_mesh_bearer_platform_ctrl_t *const *p_ctrl*, uint16_t *handle* )

Terminate Connection. API to terminate connection.

Implements rm_mesh_bearer_platform_api_t::disconnect.

Example:

```
/* Terminate Connection. */
   err = RM_MESH_BEARER_PLATFORM_Disconnect(&g_mesh_bearer_platform0_ctrl, handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Config Client (rm_mesh_config_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_CONFIG_CLT_Open (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_config_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_CONFIG_CLT_Close (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_SetServer (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t server_addr, uint8_t *p_dev_key) |
| fsp_err_t | RM_MESH_CONFIG_CLT_SendReliablePdu (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_CONFIG_CLT_BeaconGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_BeaconSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_CompositionDataGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_DefaultTtlGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_DefaultTtlSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_GattProxyGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_GattProxySet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_RelayGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_RelaySet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

| | |
|---|---|
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelPublicationGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelPublicationSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelPublicationVaddrSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionAdd (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrAdd (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionDelete (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrDelete (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionOverwrite (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrOverwrite (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelSubscriptionDeleteAll (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_SigModelSubscriptionGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_VendorModelSubscriptionGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NetkeyAdd (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

| | |
|---|---|
| fsp_err_t | RM_MESH_CONFIG_CLT_NetkeyUpdate (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NetkeyDelete (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NetkeyGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_AppkeyAdd (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_AppkeyUpdate (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_AppkeyDelete (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_AppkeyGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NodeIdentityGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NodeIdentitySet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelAppBind (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_ModelAppUnbind (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_SigModelAppGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_VendorModelAppGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NodeReset (rm_ble_mesh_config_client_ctrl_t |

| | |
|---|---|
| | *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_FriendGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_FriendSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_KeyrefreshPhaseGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_KeyrefreshPhaseSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_HeartbeatPublicationGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_HeartbeatPublicationSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_HeartbeatSubscriptionGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_HeartbeatSubscriptionSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_LpnPolltimeoutGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NetworkTransmitGet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_CONFIG_CLT_NetworkTransmitSet (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Config Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_config_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_config_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Config Client (rm_mesh_config_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Config Client (rm_mesh_config_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh config client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_config_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_config_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_config_clt_instance_ctrl_t

| struct rm_mesh_config_clt_instance_ctrl_t |
|---|
| BLE mesh config instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_CONFIG_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_CONFIG_CLT_Open()

fsp_err_t RM_MESH_CONFIG_CLT_Open ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_config_client_cfg_t const *const *p_cfg* )

Open Configuration Client middleware. This is to initialize Configuration Client model and to register with Access layer.

Implements rm_ble_mesh_config_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_CONFIG_CLT_Open(&g_mesh_config_clt0_ctrl, &g_mesh_config_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_CONFIG_CLT_Close()

fsp_err_t RM_MESH_CONFIG_CLT_Close ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

Close Configuration Client middleware.

Implements rm_ble_mesh_config_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_CONFIG_CLT_Close(&g_mesh_config_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_CONFIG_CLT_SetServer()

fsp_err_t RM_MESH_CONFIG_CLT_SetServer ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *server_addr*, uint8_t * *p_dev_key* )

This is to sets the information about server which is to be configured.

Implements rm_ble_mesh_config_client_api_t::setServer.

Example:

```
/* Sets the information about server which is to be configured. */
   err = RM_MESH_CONFIG_CLT_SetServer(&g_mesh_config_clt0_ctrl, server_addr,
&dev_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |

### ◆ RM_MESH_CONFIG_CLT_SendReliablePdu()

fsp_err_t RM_MESH_CONFIG_CLT_SendReliablePdu ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_config_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_CONFIG_CLT_SendReliablePdu(&g_mesh_config_clt0_ctrl, req_opcode,
p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_BeaconGet()

fsp_err_t RM_MESH_CONFIG_CLT_BeaconGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Beacon Get is an acknowledged message used to get the current Secure Network Beacon state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_BeaconSet()

fsp_err_t RM_MESH_CONFIG_CLT_BeaconSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Beacon Set is an acknowledged message used to set the current Secure Network Beacon state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_beacon_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_CompositionDataGet()

fsp_err_t RM_MESH_CONFIG_CLT_CompositionDataGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Composition Data Get is an acknowledged message used to read one page of the Composition Data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_composition_data_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_DefaultTtlGet()

fsp_err_t RM_MESH_CONFIG_CLT_DefaultTtlGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

Config Default TTL Get is an acknowledged message used to get the current Default TTL state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_DefaultTtlSet()

fsp_err_t RM_MESH_CONFIG_CLT_DefaultTtlSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Default TTL Set is an acknowledged message used to set the Default TTL state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_default_ttl_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_GattProxyGet()

fsp_err_t RM_MESH_CONFIG_CLT_GattProxyGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config GATT Proxy Get is an acknowledged message used to get the GATT Proxy state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_GattProxySet()

fsp_err_t RM_MESH_CONFIG_CLT_GattProxySet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config GATT Proxy Set is an acknowledged message used to set the GATT Proxy state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_gatt_proxy_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_RelayGet()

fsp_err_t RM_MESH_CONFIG_CLT_RelayGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Relay Get is an acknowledged message used to get the current Relay and Relay Retransmit states of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_RelaySet()

fsp_err_t RM_MESH_CONFIG_CLT_RelaySet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Relay Set is an acknowledged message used to set the current Relay and Relay Retransmit states of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_relay_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelPublicationGet()

fsp_err_t RM_MESH_CONFIG_CLT_ModelPublicationGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Publication Get is an acknowledged message used to get the publish address and parameters of an outgoing message that originates from a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_publication_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelPublicationSet()

fsp_err_t RM_MESH_CONFIG_CLT_ModelPublicationSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Publication Set is an acknowledged message used to set the Model Publication state of an outgoing message that originates from a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_publication_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelPublicationVaddrSet()

fsp_err_t RM_MESH_CONFIG_CLT_ModelPublicationVaddrSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Publication Set is an acknowledged message used to set the Model Publication state of an outgoing message that originates from a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_publication_virtual_address_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionAdd()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionAdd ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Add is an acknowledged message used to add an address to a Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_add_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrAdd()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrAdd ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Add is an acknowledged message used to add an address to a Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_virtual_address_add_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionDelete()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionDelete ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Delete is an acknowledged message used to delete a subscription address from the Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_delete_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrDelete()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrDelete ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Delete is an acknowledged message used to delete a subscription address from the Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_virtual_address_delete_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionOverwrite()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionOverwrite ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Overwrite is an acknowledged message used to discard the Subscription List and add an address to the cleared Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_overwrite_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrOverwrite()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionVaddrOverwrite ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Overwrite is an acknowledged message used to discard the Subscription List and add an address to the cleared Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_virtual_address_overwrite_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelSubscriptionDeleteAll()

fsp_err_t RM_MESH_CONFIG_CLT_ModelSubscriptionDeleteAll ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model Subscription Delete All is an acknowledged message used to discard the Subscription List of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_subscription_delete_all_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_SigModelSubscriptionGet()

fsp_err_t RM_MESH_CONFIG_CLT_SigModelSubscriptionGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config SIG Model Subscription Get is an acknowledged message used to get the list of subscription addresses of a model within the element. This message is only for SIG Models.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_sig_model_subscription_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_VendorModelSubscriptionGet()

fsp_err_t RM_MESH_CONFIG_CLT_VendorModelSubscriptionGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config SIG Model Subscription Get is an acknowledged message used to get the list of subscription addresses of a model within the element. This message is only for Vendor Models.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_vendor_model_subscription_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetkeyAdd()

fsp_err_t RM_MESH_CONFIG_CLT_NetkeyAdd ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config NetKey Add is an acknowledged message used to add a NetKey to a NetKey List on a node. The added NetKey is then used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_netkey_add_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetkeyUpdate()

fsp_err_t RM_MESH_CONFIG_CLT_NetkeyUpdate ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config NetKey Update is an acknowledged message used to update a NetKey on a node. The updated NetKey is then used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends, as defined by the Key Refresh procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_netkey_update_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetkeyDelete()

fsp_err_t RM_MESH_CONFIG_CLT_NetkeyDelete ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config NetKey Delete is an acknowledged message used to delete a NetKey on a NetKey List from a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_netkey_delete_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetkeyGet()

fsp_err_t RM_MESH_CONFIG_CLT_NetkeyGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config NetKey Get is an acknowledged message used to report all NetKeys known to the node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_AppkeyAdd()

fsp_err_t RM_MESH_CONFIG_CLT_AppkeyAdd ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config AppKey Add is an acknowledged message used to add an AppKey to the AppKey List on a node and bind it to the NetKey identified by NetKeyIndex. The added AppKey can be used by the node only as a pair with the specified NetKey. The AppKey is used to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_appkey_add_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_AppkeyUpdate()

fsp_err_t RM_MESH_CONFIG_CLT_AppkeyUpdate ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config AppKey Update is an acknowledged message used to update an AppKey value on the AppKey List on a node. The updated AppKey is used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends, as defined by the Key Refresh procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_appkey_update_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_AppkeyDelete()

fsp_err_t RM_MESH_CONFIG_CLT_AppkeyDelete ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config AppKey Delete is an acknowledged message used to delete an AppKey from the AppKey List on a node

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_appkey_delete_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_AppkeyGet()

fsp_err_t RM_MESH_CONFIG_CLT_AppkeyGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The AppKey Get is an acknowledged message used to report all AppKeys bound to the NetKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_appkey_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NodeIdentityGet()

fsp_err_t RM_MESH_CONFIG_CLT_NodeIdentityGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Node Identity Get is an acknowledged message used to get the current Node Identity state for a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_node_id_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NodeIdentitySet()

fsp_err_t RM_MESH_CONFIG_CLT_NodeIdentitySet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Node Identity Set is an acknowledged message used to set the current Node Identity state for a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_node_id_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelAppBind()

fsp_err_t RM_MESH_CONFIG_CLT_ModelAppBind ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model App Bind is an acknowledged message used to bind an AppKey to a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_app_bind_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_ModelAppUnbind()

fsp_err_t RM_MESH_CONFIG_CLT_ModelAppUnbind ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Model App Unbind is an acknowledged message used to remove the binding between an AppKey and a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_model_app_unbind_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_SigModelAppGet()

fsp_err_t RM_MESH_CONFIG_CLT_SigModelAppGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config SIG Model App Get is an acknowledged message used to request report of all AppKeys bound to the SIG Model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_sig_model_app_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_VendorModelAppGet()

fsp_err_t RM_MESH_CONFIG_CLT_VendorModelAppGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Vendor Model App Get is an acknowledged message used to request report of all AppKeys bound to the Vendor Model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_vendor_model_app_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NodeReset()

fsp_err_t RM_MESH_CONFIG_CLT_NodeReset ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Node Reset is an acknowledged message used to reset a node (other than a Provisioner) and remove it from the network.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_FriendGet()

fsp_err_t RM_MESH_CONFIG_CLT_FriendGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Friend Get is an acknowledged message used to get the current Friend state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_FriendSet()

fsp_err_t RM_MESH_CONFIG_CLT_FriendSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Friend Set is an acknowledged message used to set the Friend state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_friend_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_KeyrefreshPhaseGet()

fsp_err_t RM_MESH_CONFIG_CLT_KeyrefreshPhaseGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Key Refresh Phase Get is an acknowledged message used to get the current Key Refresh Phase state of the identified network key.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_key_refresh_phase_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_KeyrefreshPhaseSet()

fsp_err_t RM_MESH_CONFIG_CLT_KeyrefreshPhaseSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Key Refresh Phase Set is an acknowledged message used to set the current Key Refresh Phase state of the identified network key.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_key_refresh_phase_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_HeartbeatPublicationGet()

fsp_err_t RM_MESH_CONFIG_CLT_HeartbeatPublicationGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Heartbeat Publication Get is an acknowledged message used to get the current Heartbeat Publication state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_HeartbeatPublicationSet()

fsp_err_t RM_MESH_CONFIG_CLT_HeartbeatPublicationSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Heartbeat Publication Set is an acknowledged message used to set the current Heartbeat Publication state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_heartbeat_publication_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_HeartbeatSubscriptionGet()

fsp_err_t RM_MESH_CONFIG_CLT_HeartbeatSubscriptionGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Heartbeat Subscription Get is an acknowledged message used to get the current Heartbeat Subscription state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_HeartbeatSubscriptionSet()

fsp_err_t RM_MESH_CONFIG_CLT_HeartbeatSubscriptionSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Heartbeat Publication Set is an acknowledged message used to set the current Heartbeat Subscription state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_heartbeat_subscription_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_LpnPolltimeoutGet()

fsp_err_t RM_MESH_CONFIG_CLT_LpnPolltimeoutGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Low Power Node PollTimeout Get is an acknowledged message used to get the current value of PollTimeout timer of the Low Power node within a Friend node. The message is sent to a Friend node that has claimed to be handling messages by sending ACKs On Behalf Of (OBO) the indicated Low Power node. This message should only be sent to a node that has the Friend feature supported and enabled.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_low_power_node_polling_timeout_get_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetworkTransmitGet()

fsp_err_t RM_MESH_CONFIG_CLT_NetworkTransmitGet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*)

The Config Network Transmit Get is an acknowledged message used to get the current Network Transmit state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_CONFIG_CLT_NetworkTransmitSet()

fsp_err_t RM_MESH_CONFIG_CLT_NetworkTransmitSet ( rm_ble_mesh_config_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Config Network Transmit Set is an acknowledged message used to set the current Network Transmit state of a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_config_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_config_clt_network_transmit_set_parameter_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Config Server (rm_mesh_config_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_CONFIG_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_CONFIG_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_CONFIG_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Config Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_config_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_config_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Config Server (rm_mesh_config_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Config Server (rm_mesh_config_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh config client ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic admin property server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_config_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_config_srv_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_config_srv_instance_ctrl_t

struct rm_mesh_config_srv_instance_ctrl_t

BLE mesh config instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_CONFIG_SRV_Open() is called.

## Function Documentation

### ◆ RM_MESH_CONFIG_SRV_Open()

fsp_err_t RM_MESH_CONFIG_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize configuration server model. This is to initialize configuration server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_CONFIG_SRV_Open(&g_mesh_config_srv0_ctrl, &g_mesh_config_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_CONFIG_SRV_Close()

fsp_err_t RM_MESH_CONFIG_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate configuration server model. This is to terminate configuration server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_CONFIG_SRV_Close(&g_mesh_config_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_CONFIG_SRV_StateUpdate()

fsp_err_t RM_MESH_CONFIG_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_CONFIG_SRV_StateUpdate(&g_mesh_config_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is unsupported. |

## BLE Mesh Network Generic Admin Property Server (rm_mesh_generic_admin_prop_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ADMIN_PROP_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ADMIN_PROP_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ADMIN_PROP_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Admin Property Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_admin_prop_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_admin_prop_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Admin Property Server (rm_mesh_generic_admin_prop_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Admin Property Server (rm_mesh_generic_admin_prop_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic admin property server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic admin property server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_admin_prop_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_admin_prop_srv_info_t |

| | |
|---|---|
| struct | rm_mesh_generic_admin_prop_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_admin_prop_srv_info_t

| struct rm_mesh_generic_admin_prop_srv_info_t | | |
|---|---|---|
| Generic Admin Property is a state representing a device property of an element that can be read or written | | |
| Data Fields | | |
| uint16_t | property_id | Admin Property ID field is a 2-octet Assigned Number value referencing a device property |
| uint8_t | user_access | Admin User Access field is an enumeration indicating whether the device property can be read or written as a Generic User Property |
| uint8_t * | property_value | Admin Property Value field is a conditional field |
| uint16_t | property_value_len | |

### ◆ rm_mesh_generic_admin_prop_srv_instance_ctrl_t

| struct rm_mesh_generic_admin_prop_srv_instance_ctrl_t |
|---|
| BLE mesh generic admin prop instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_ADMIN_PROP_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_ADMIN_PROP_SRV_Open()

fsp_err_t RM_MESH_GENERIC_ADMIN_PROP_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Admin_Property Server model. This is to initialize Generic_Admin_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err =
RM_MESH_GENERIC_ADMIN_PROP_SRV_Open(&g_mesh_generic_admin_prop_srv0_ctrl,
&g_mesh_generic_admin_prop_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_ADMIN_PROP_SRV_Close()

fsp_err_t RM_MESH_GENERIC_ADMIN_PROP_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Admin_Property Server model. This is to terminate Generic_Admin_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_ADMIN_PROP_SRV_Close(&g_mesh_generic_admin_prop_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_ADMIN_PROP_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_ADMIN_PROP_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err =
RM_MESH_GENERIC_ADMIN_PROP_SRV_StateUpdate(&g_mesh_generic_admin_prop_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Battery Client (rm_mesh_generic_battery_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_BATTERY_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_CLT_Close |

| | |
|---|---|
| | (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Battery Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_battery_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_battery_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Battery Client (rm_mesh_generic_battery_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Battery Client (rm_mesh_generic_battery_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic battery client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_battery_clt0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_battery_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_battery_clt_instance_ctrl_t

| struct rm_mesh_generic_battery_clt_instance_ctrl_t |
|---|
| BLE mesh generic battery instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_BATTERY_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_BATTERY_CLT_Open()

fsp_err_t RM_MESH_GENERIC_BATTERY_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic Battery Client middleware. This is to initialize Generic_Battery Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_BATTERY_CLT_Open(&g_mesh_generic_battery_clt0_ctrl,
&g_mesh_generic_battery_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_BATTERY_CLT_Close()

fsp_err_t RM_MESH_GENERIC_BATTERY_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic Battery Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_BATTERY_CLT_Close(&g_mesh_generic_battery_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_BATTERY_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_BATTERY_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Battery client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic battery client model. */
   err =
RM_MESH_GENERIC_BATTERY_CLT_GetModelHandle(&g_mesh_generic_battery_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_BATTERY_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_BATTERY_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err =
RM_MESH_GENERIC_BATTERY_CLT_SendReliablePdu(&g_mesh_generic_battery_clt0_ctrl,
                                            req_opcode,
                                            p_parameter,
                                            rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_BATTERY_CLT_Get()

fsp_err_t RM_MESH_GENERIC_BATTERY_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Battery Get message is an acknowledged message used to get the Generic Battery state of an element. The response to the Generic Battery Get message is a Generic Battery Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_battery_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Battery Server (rm_mesh_generic_battery_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_BATTERY_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_BATTERY_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Battery Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_battery_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_battery_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Battery Server (rm_mesh_generic_battery_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Battery Server (rm_mesh_generic_battery_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic battery server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic battery server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_battery_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_battery_srv_info_t |
| struct | rm_mesh_generic_battery_srv_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_generic_battery_srv_info_t

| struct rm_mesh_generic_battery_srv_info_t |
|---|

Generic Battery state is a set of four values representing the state of a battery

| Data Fields | | |
|---|---|---|
| uint8_t | generic_battery_level | Generic Battery Level state is a value ranging from 0 percent through 100 percent |
| uint32_t | generic_battery_time_to_discharge | Generic Battery Time to Discharge state is a 24-bit unsigned value ranging from 0 through 0xFFFFFF |
| uint32_t | generic_battery_time_to_charge | Generic Battery Time to Charge state is a 24-bit unsigned value ranging from 0 through 0xFFFFFF |
| uint8_t | generic_battery_flags | Generic Battery Flags state is a concatenation of four 2-bit bit fields: Presence, Indicator, Charging, and Serviceability |

◆ **rm_mesh_generic_battery_srv_instance_ctrl_t**

struct rm_mesh_generic_battery_srv_instance_ctrl_t

BLE mesh generic_battery instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_BATTERY_SRV_Open() is called.

## Function Documentation

### ◆ RM_MESH_GENERIC_BATTERY_SRV_Open()

fsp_err_t RM_MESH_GENERIC_BATTERY_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Battery Server model. This is to initialize Generic_Battery Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_BATTERY_SRV_Open(&g_mesh_generic_battery_srv0_ctrl,
&g_mesh_generic_battery_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_BATTERY_SRV_Close()

fsp_err_t RM_MESH_GENERIC_BATTERY_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Battery Server model. This is to terminate Generic_Battery Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_BATTERY_SRV_Close(&g_mesh_generic_battery_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_BATTERY_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_BATTERY_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_BATTERY_SRV_StateUpdate(&g_mesh_generic_battery_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Client Property Server (rm_mesh_generic_client_prop_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_CLIENT_PROP_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_CLIENT_PROP_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_CLIENT_PROP_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Client Property Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_client_prop_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_client_prop_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Client Property Server (rm_mesh_generic_client_prop_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Client Property Server (rm_mesh_generic_client_prop_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic client property server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic client property server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_client_prop_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_client_prop_srv_instance_ctrl_t |

### Data Structure Documentation

◆ **rm_mesh_generic_client_prop_srv_instance_ctrl_t**

| struct rm_mesh_generic_client_prop_srv_instance_ctrl_t |
| --- |
| BLE mesh generic client prop instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_CLIENT_PROP_SRV_Open() is called. |

### Function Documentation

#### ◆ RM_MESH_GENERIC_CLIENT_PROP_SRV_Open()

fsp_err_t RM_MESH_GENERIC_CLIENT_PROP_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Client_Property Server model. This is to initialize Generic_Client_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_CLIENT_PROP_SRV_Open(&g_mesh_generic_client_prop_srv0_ctrl,
                                              &g_mesh_generic_client_prop_srv0_cfg);
```

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_CLIENT_PROP_SRV_Close()

fsp_err_t RM_MESH_GENERIC_CLIENT_PROP_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Client_Property Server model. This is to terminate Generic_Client_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err =
RM_MESH_GENERIC_CLIENT_PROP_SRV_Close(&g_mesh_generic_client_prop_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_CLIENT_PROP_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_CLIENT_PROP_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err =
RM_MESH_GENERIC_CLIENT_PROP_SRV_StateUpdate(&g_mesh_generic_client_prop_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Default Transition Time Client (rm_mesh_generic_dtt_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_Close |

| | |
|---|---|
| | (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_DTT_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Default Transition Time Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_dtt_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_dtt_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Default Transition Time Client (rm_mesh_generic_dtt_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Default Transition Time Client (rm_mesh_generic_dtt_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic default transition time |

| | | | |
|---|---|---|---|
| | | | client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_dtt_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_dtt_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_dtt_clt_instance_ctrl_t

struct rm_mesh_generic_dtt_clt_instance_ctrl_t

BLE mesh generic dtt instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_DTT_CLT_Open() is called.

## Function Documentation

### ◆ RM_MESH_GENERIC_DTT_CLT_Open()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic Default Transition Time Client middleware. This is to initialize Generic_Default_Transition_Time Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_DTT_CLT_Open(&g_mesh_generic_dtt_clt0_ctrl,
&g_mesh_generic_dtt_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_DTT_CLT_Close()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic Default Transition Time Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_DTT_CLT_Close(&g_mesh_generic_dtt_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_DTT_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Default_Transition_Time client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic default transition time client model. */
   err = RM_MESH_GENERIC_DTT_CLT_GetModelHandle(&g_mesh_generic_dtt_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_DTT_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands */
   err = RM_MESH_GENERIC_DTT_CLT_SendReliablePdu(&g_mesh_generic_dtt_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_DTT_CLT_Get()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Default Transition Time Get is an acknowledged message used to get the Generic Default Transition Time state of an element. The response to the Generic Default Transition Time Get message is a Generic Default Transition Time Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_dtt_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_DTT_CLT_Set()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Default Transition Time Set is an acknowledged message used to set the Generic Default Transition Time state of an element. The response to the Generic Default Transition Time Set message is a Generic Default Transition Time Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_dtt_clt control block. |
| [in] | p_parameter | Pointer to transition time. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_DTT_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_DTT_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Default Transition Time Set Unacknowledged is an unacknowledged message used to set the Generic Default Transition Time state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_dtt_clt control block. |
| [in] | p_parameter | Pointer to transition time. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Default Transition Time Server (rm_mesh_generic_dtt_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_DTT_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_DTT_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_DTT_SRV_GetTime |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_mesh_generic_dtt_srv_transtion_time_info_t *const p_info) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Default Transition Time Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_dtt_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_dtt_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Default Transition Time Server (rm_mesh_generic_dtt_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Default Transition Time Server (rm_mesh_generic_dtt_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Tmeout Occurs | Name must Be a Valid C Symbol | NULL | Callback provided when mesh generic default transition time server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic default transition time server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_dtt_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_dtt_srv_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_generic_dtt_srv_instance_ctrl_t

| struct rm_mesh_generic_dtt_srv_instance_ctrl_t |
| --- |
| BLE mesh generic dtt instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_DTT_SRV_Open() is called. |

### Function Documentation

#### ◆ RM_MESH_GENERIC_DTT_SRV_Open()

fsp_err_t RM_MESH_GENERIC_DTT_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Default_Transition_Time Server model. This is to initialize Generic_Default_Transition_Time Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_DTT_SRV_Open(&g_mesh_generic_dtt_srv0_ctrl,
&g_mesh_generic_dtt_srv0_cfg);
```

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_DTT_SRV_Close()

fsp_err_t RM_MESH_GENERIC_DTT_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Default_Transition_Time Server model. This is to terminate Generic_Default_Transition_Time Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_DTT_SRV_Close(&g_mesh_generic_dtt_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_DTT_SRV_GetTime()

fsp_err_t RM_MESH_GENERIC_DTT_SRV_GetTime ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_mesh_generic_dtt_srv_transtion_time_info_t *const *p_info* )

API to get default transition time. This is to get default transition time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_dtt_srv control block. |
| [in] | p_info | rm_mesh_generic_dtt_srv status information. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |

## BLE Mesh Network Generic Level Client (rm_mesh_generic_level_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_DeltaSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_DeltaSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_MoveSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_CLT_MoveSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Level Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_level_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_level_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Level Client (rm_mesh_generic_level_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Level Client (rm_mesh_generic_level_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic level client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_level_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_level_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_generic_level_clt_instance_ctrl_t

| struct rm_mesh_generic_level_clt_instance_ctrl_t |
|---|
| BLE mesh generic level instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_LEVEL_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_GENERIC_LEVEL_CLT_Open()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Level Client middleware. This is to initialize Generic_Level Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_LEVEL_CLT_Open(&g_mesh_generic_level_clt0_ctrl,
&g_mesh_generic_level_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_Close()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Level Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_LEVEL_CLT_Close(&g_mesh_generic_level_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Level client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle().

Example:

```
/* Get the handle of generic level client model. */
   err = RM_MESH_GENERIC_LEVEL_CLT_GetModelHandle(&g_mesh_generic_level_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err =
RM_MESH_GENERIC_LEVEL_CLT_SendReliablePdu(&g_mesh_generic_level_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_Get()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Level Get is an acknowledged message used to get the Generic Level state of an element. The response to the Generic Level Get message is a Generic Level Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_Set()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Level Set is an acknowledged message used to set the Generic Level state of an element to a new absolute value. The response to the Generic Level Set message is a Generic Level Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Level Set Unacknowledged is an unacknowledged message used to set the Generic Level state of an element to a new absolute value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_DeltaSet()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_DeltaSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Delta Set is an acknowledged message used to set the Generic Level state of an element by a relative value. The message is transactional, it supports changing the state by a cumulative value with a sequence of messages that are part of a transaction. The response to the Generic Delta Set message is a Generic Level Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_DeltaSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_DeltaSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Delta Set Unacknowledged is an unacknowledged message used to set the Generic Level state of an element by a relative value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_MoveSet()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_MoveSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Move Set is an acknowledged message used to start a process of changing the Generic Level state of an element with a defined transition speed. The response to the Generic Move Set message is a Generic Level Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_CLT_MoveSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_LEVEL_CLT_MoveSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Move Set Unacknowledged is an unacknowledged message used to start a process of changing the Generic Level state of an element with a defined transition speed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_level_clt control block. |
| [in] | p_parameter | Pointer to Generic Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Level Server (rm_mesh_generic_level_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LEVEL_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_LEVEL_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LEVEL_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Level Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_level_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_level_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Level Server (rm_mesh_generic_level_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Level Server (rm_mesh_generic_level_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided Shen Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic level server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic level server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_level_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_level_srv_info_t |
| struct | rm_mesh_generic_level_srv_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_generic_level_srv_info_t

| struct rm_mesh_generic_level_srv_info_t | | |
|---|---|---|
| Generic Level state is a 16-bit signed integer (2-s complement) representing the state of an element | | |
| Data Fields | | |
| uint16_t | level | Generic Level |
| uint32_t | delta_level | Delta Level |
| uint16_t | move_level | Move Level |
| uint8_t | tid | |
| uint8_t | transition_time | |
| uint8_t | delay | |
| uint16_t | target_level | Target Level |

### ◆ rm_mesh_generic_level_srv_instance_ctrl_t

| struct rm_mesh_generic_level_srv_instance_ctrl_t |
|---|
| BLE mesh generic level instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_LEVEL_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_LEVEL_SRV_Open()

fsp_err_t RM_MESH_GENERIC_LEVEL_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Level Server model. This is to initialize Generic_Level Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_LEVEL_SRV_Open(&g_mesh_generic_level_srv0_ctrl,
&g_mesh_generic_level_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_LEVEL_SRV_Close()

fsp_err_t RM_MESH_GENERIC_LEVEL_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Level Server model. This is to terminate Generic_Level Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_LEVEL_SRV_Close(&g_mesh_generic_level_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_LEVEL_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_LEVEL_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_BATTERY_SRV_StateUpdate(&g_mesh_generic_battery_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Location Client (rm_mesh_generic_loc_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationGlobalGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationGlobalSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationGlobalSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationLocalGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationLocalSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_LOC_CLT_LocationLocalSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Location Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_loc_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_loc_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic**

## Location Client (rm_mesh_generic_loc_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Location Client (rm_mesh_generic_loc_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic location client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_loc_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_loc_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_generic_loc_clt_instance_ctrl_t

| struct rm_mesh_generic_loc_clt_instance_ctrl_t |
|---|
| BLE mesh generic loc instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_LOC_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_GENERIC_LOC_CLT_Open()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Location Client middleware. This is to initialize Generic_Location Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_LOC_CLT_Open(&g_mesh_generic_loc_clt0_ctrl,
&g_mesh_generic_loc_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_Close()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Location Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_LOC_CLT_Close(&g_mesh_generic_loc_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_LOC_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Location client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic_location client model. */
   err = RM_MESH_GENERIC_LOC_CLT_GetModelHandle(&g_mesh_generic_loc_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_LOC_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands */
   err = RM_MESH_GENERIC_LOC_CLT_SendReliablePdu(&g_mesh_generic_loc_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationGlobalGet()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationGlobalGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Location Global Get message is an acknowledged message used to get the selected fields of the Generic Location state of an element. The response to the Generic Location Global Get message is a Generic Location Global Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationGlobalSet()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationGlobalSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Location Global Set is an acknowledged message used to set the selected fields of the Generic Location state of an element. The response to the Generic Location Global Set message is a Generic Location Global Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |
| [in] | p_parameter | Pointer to Generic Location Global Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationGlobalSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationGlobalSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Location Global Set Unacknowledged is an unacknowledged message used to set the selected fields of the Generic Location state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |
| [in] | p_parameter | Pointer to Generic Location Global Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationLocalGet()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationLocalGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Location Local Get message is an acknowledged message used to get the selected fields of the Generic Location state of an element. The response to the Generic Location Local Get message is a Generic Location Local Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationLocalSet()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationLocalSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Location Local Set is an acknowledged message used to set the selected fields of the Generic Location state of an element. The response to the Generic Location Local Set message is a Generic Location Local Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |
| [in] | p_parameter | Pointer to Generic Location Local Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_CLT_LocationLocalSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_LOC_CLT_LocationLocalSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Location Local Set Unacknowledged is an unacknowledged message used to set the selected fields of the Generic Location state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_clt control block. |
| [in] | p_parameter | Pointer to Generic Location Local Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Location Server (rm_mesh_generic_loc_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LOC_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_LOC_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_LOC_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_GENERIC_LOC_SRV_SetupServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Location Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_loc_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_loc_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Location Server (rm_mesh_generic_loc_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Location Server (rm_mesh_generic_loc_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic location server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic location server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_loc_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_loc_srv_global_info_t |
| struct | rm_mesh_generic_loc_srv_local_info_t |

| | |
|---|---|
| struct | rm_mesh_generic_loc_srv_state_info_t |
| struct | rm_mesh_generic_loc_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_loc_srv_global_info_t

| struct rm_mesh_generic_loc_srv_global_info_t | | |
|---|---|---|
| Generic Global Location state defines location information of an element | | |
| Data Fields | | |
| uint32_t | global_latitude | Global Coordinates (Latitude) |
| uint32_t | global_longitude | Global Coordinates (Longitude) |
| uint16_t | global_altitude | Global Altitude |

### ◆ rm_mesh_generic_loc_srv_local_info_t

| struct rm_mesh_generic_loc_srv_local_info_t | | |
|---|---|---|
| Generic Local Location state defines location information of an element | | |
| Data Fields | | |
| uint16_t | local_north | Local Coordinates (North) |
| uint16_t | local_east | Local Coordinates (East) |
| uint16_t | local_altitude | Local Altitude |
| uint8_t | floor_number | Floor Number |
| uint16_t | uncertainty | Uncertainty |

### ◆ rm_mesh_generic_loc_srv_state_info_t

| struct rm_mesh_generic_loc_srv_state_info_t | | |
|---|---|---|
| Generic Location state is a composite state that includes a Generic Location Global state and a Generic Location Local state | | |
| Data Fields | | |
| rm_mesh_generic_loc_srv_global_info_t | global_location | Global Location |
| rm_mesh_generic_loc_srv_local_info_t | local_location | Local Location |

### ◆ rm_mesh_generic_loc_srv_instance_ctrl_t

| struct rm_mesh_generic_loc_srv_instance_ctrl_t |
|---|
| BLE mesh generic loc instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_LOC_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_LOC_SRV_Open()

fsp_err_t RM_MESH_GENERIC_LOC_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Location Server model and to initialize Generic_Location_Setup Server model. This is to initialize Generic_Location Server model and to register with Access layer. And this is to initialize Generic_Location_Setup Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_LOC_SRV_Open(&g_mesh_generic_loc_srv0_ctrl,
&g_mesh_generic_loc_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_LOC_SRV_Close()

fsp_err_t RM_MESH_GENERIC_LOC_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Location Server model. This is to terminate Generic_Location Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_LOC_SRV_Close(&g_mesh_generic_loc_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_LOC_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_LOC_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_LOC_SRV_StateUpdate(&g_mesh_generic_loc_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_LOC_SRV_SetupServerStateUpdate()

fsp_err_t RM_MESH_GENERIC_LOC_SRV_SetupServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_loc_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Manufacturer Property Server (rm_mesh_generic_mfr_prop_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_MFR_PROP_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_MFR_PROP_SRV_Close |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_MFR_PROP_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Manufacturer Property Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_mfr_prop_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_mfr_prop_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Manufacturer Property Server (rm_mesh_generic_mfr_prop_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Manufacturer Property Server (rm_mesh_generic_mfr_prop_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic manufacturer property server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must be a Valid C Symbol | NULL | Callback provided when mesh generic manufacturer property server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_mfr_prop_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_mfr_prop_srv_info_t |
| struct | rm_mesh_generic_mfr_prop_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_mfr_prop_srv_info_t

| struct rm_mesh_generic_mfr_prop_srv_info_t | | |
|---|---|---|
| Generic Manufacturer Property is a state representing a device property of an element that is programmed by a manufacturer and can be read | | |
| Data Fields | | |
| uint16_t | property_id | Manufacturer Property ID field is a 2-octet Assigned Number value that references a device property |
| uint8_t | user_access | Manufacturer User Access field is an enumeration indicating whether or not the device property can be read as a Generic User Property |
| uint8_t * | property_value | Manufacturer Property Value field is a conditional field |
| uint16_t | property_value_len | |

### ◆ rm_mesh_generic_mfr_prop_srv_instance_ctrl_t

| struct rm_mesh_generic_mfr_prop_srv_instance_ctrl_t |
|---|
| BLE mesh generic mfr prop instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_MFR_PROP_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_MFR_PROP_SRV_Open()

fsp_err_t RM_MESH_GENERIC_MFR_PROP_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Manufacturer_Property Server model. This is to initialize Generic_Manufacturer_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_MFR_PROP_SRV_Open(&g_mesh_generic_mfr_prop_srv0_ctrl,
&g_mesh_generic_mfr_prop_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_MFR_PROP_SRV_Close()

fsp_err_t RM_MESH_GENERIC_MFR_PROP_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Manufacturer_Property Server model. This is to terminate Generic_Manufacturer_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_MFR_PROP_SRV_Close(&g_mesh_generic_mfr_prop_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_MFR_PROP_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_MFR_PROP_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err =
RM_MESH_GENERIC_MFR_PROP_SRV_StateUpdate(&g_mesh_generic_mfr_prop_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic On Off Client (rm_mesh_generic_on_off_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic On Off Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_on_off_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_on_off_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic On Off Client (rm_mesh_generic_on_off_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic On Off Client (rm_mesh_generic_on_off_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic on off client ISR occurs |

| | | | |
|---|---|---|---|
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_on_off_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_on_off_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_on_off_clt_instance_ctrl_t

struct rm_mesh_generic_on_off_clt_instance_ctrl_t

BLE mesh generic on off instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_ON_OFF_CLT_Open() is called.

## Function Documentation

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_Open()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Onoff Client middleware. This is to initialize Generic_Onoff Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_ON_OFF_CLT_Open(&g_mesh_generic_on_off_clt0_ctrl,
&g_mesh_generic_on_off_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_Close()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Onoff Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_ON_OFF_CLT_Close(&g_mesh_generic_on_off_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Onoff client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic on_off client model. */
   err = RM_MESH_GENERIC_ON_OFF_CLT_GetModelHandle(&g_mesh_generic_on_off_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err =
RM_MESH_GENERIC_ON_OFF_CLT_SendReliablePdu(&g_mesh_generic_on_off_clt0_ctrl,
                                           req_opcode,
                                           p_parameter,
                                           rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_Get()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Generic OnOff Get is an acknowledged message used to get the Generic OnOff state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_on_off_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_Set()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Generic OnOff Set is an acknowledged message used to get the Generic OnOff state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_on_off_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_generic_on_off_set_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_ON_OFF_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_ON_OFF_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Generic OnOff Set is an unacknowledged message used to get the Generic OnOff state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_on_off_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_generic_on_off_set_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic On Off Server (rm_mesh_generic_on_off_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_SRV_Close |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_ON_OFF_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic On Off Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_on_off_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_on_off_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic On Off Server (rm_mesh_generic_on_off_srv)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic On Off Server (rm_mesh_generic_on_off_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic on off server ISR occurs |
| Callback Provided When an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic on off server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_on_off_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_on_off_srv_info_t |
| struct | rm_mesh_generic_on_off_srv_instance_ctrl_t |

**Data Structure Documentation**

### ◆ rm_mesh_generic_on_off_srv_info_t

| struct rm_mesh_generic_on_off_srv_info_t | | |
|---|---|---|
| Generic OnOff state is a Boolean value that represents the state of an element | | |
| Data Fields | | |
| uint8_t | onoff | Generic On/Off |
| uint8_t | target_onoff | Target On/Off - Used in response path |
| uint8_t | last_onoff | Last On/Off |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_generic_on_off_srv_instance_ctrl_t

| struct rm_mesh_generic_on_off_srv_instance_ctrl_t |
|---|
| BLE mesh generic on off instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_ON_OFF_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_ON_OFF_SRV_Open()

fsp_err_t RM_MESH_GENERIC_ON_OFF_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Onoff Server model. This is to initialize Generic_Onoff Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_ON_OFF_SRV_Open(&g_mesh_generic_on_off_srv0_ctrl,
&g_mesh_generic_on_off_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_ON_OFF_SRV_Close()

fsp_err_t RM_MESH_GENERIC_ON_OFF_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Onoff Server model. This is to terminate Generic_Onoff Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_ON_OFF_SRV_Close(&g_mesh_generic_on_off_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_ON_OFF_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_ON_OFF_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_ON_OFF_SRV_StateUpdate(&g_mesh_generic_on_off_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Power Level Client (rm_mesh_generic_pl_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_LevelGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_LevelSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_LevelSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_LastGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_DefaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_DefaultSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_DefaultSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_RangeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_RangeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PL_CLT_RangeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Power Level Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_pl_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_pl_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power Level Client (rm_mesh_generic_pl_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power Level Client (rm_mesh_generic_pl_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided When Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power level client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_pl_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_pl_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_pl_clt_instance_ctrl_t

| struct rm_mesh_generic_pl_clt_instance_ctrl_t |
|---|
| BLE mesh generic pl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_PL_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_PL_CLT_Open()

fsp_err_t RM_MESH_GENERIC_PL_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Power_Level Client middleware. This is to initialize Generic_Power_Level Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_PL_CLT_Open(&g_mesh_generic_pl_clt0_ctrl,
&g_mesh_generic_pl_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_Close()

fsp_err_t RM_MESH_GENERIC_PL_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Power_Level Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_PL_CLT_Close(&g_mesh_generic_pl_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_PL_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_PL_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Power_Level client model. Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic power level client model. */
   err = RM_MESH_GENERIC_PL_CLT_GetModelHandle(&g_mesh_generic_pl_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_PL_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_PL_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands. Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_GENERIC_PL_CLT_SendReliablePdu(&g_mesh_generic_pl_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_LevelGet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_LevelGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Power Level Get message is an acknowledged message used to get the Generic Power Actual state of an element. The response to the Generic Power Level Get message is a Generic Power Level Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_LevelSet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_LevelSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Level Set is an acknowledged message used to set the Generic Power Actual state of an element. The response to the Generic Power Level Set message is a Generic Power Level Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_LevelSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PL_CLT_LevelSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Level Set Unacknowledged is an unacknowledged message used to set the Generic Power Actual state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Level Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_LastGet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_LastGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Power Last Get is an acknowledged message used to get the Generic Power Last state of an element. The response to a Generic Power Last Get message is a Generic Power Last Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_DefaultGet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_DefaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Power Default Get is an acknowledged message used to get the Generic Power Default state of an element. The response to a Generic Power Default Get message is a Generic Power Default Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_DefaultSet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_DefaultSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Default Set is an acknowledged message used to set the Generic Power Default state of an element. The response to the Generic Power Default Set message is a Generic Power Default Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_DefaultSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PL_CLT_DefaultSetUnacknowledged (
rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Default Set Unacknowledged is an unacknowledged message used to set the Generic Power Default state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_RangeGet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_RangeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Power Range Get is an acknowledged message used to get the Generic Power Range state of an element. The response to the Generic Power Range Get message is a Generic Power Range Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_RangeSet()

fsp_err_t RM_MESH_GENERIC_PL_CLT_RangeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Range Set is an acknowledged message used to set the Generic Power Range state of an element. The response to the Generic Power Range Set message is a Generic Power Range Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_CLT_RangeSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PL_CLT_RangeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Power Range Set Unacknowledged is an unacknowledged message used to set the Generic Power Range state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_clt control block. |
| [in] | p_parameter | Pointer to Generic Power Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Power Level Server (rm_mesh_generic_pl_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PL_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_PL_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PL_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_GENERIC_PL_SRV_SetupServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Power Level Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_pl_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_pl_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power Level Server (rm_mesh_generic_pl_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power Level Server (rm_mesh_generic_pl_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power level server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power level server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_pl_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_pl_srv_actual_state_info_t |
| struct | rm_mesh_generic_pl_srv_last_state_info_t |

| | |
|---|---|
| struct | rm_mesh_generic_pl_srv_default_state_info_t |
| struct | rm_mesh_generic_pl_srv_range_state_info_t |
| struct | rm_mesh_generic_pl_srv_state_info_t |
| struct | rm_mesh_generic_pl_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_pl_srv_actual_state_info_t

| struct rm_mesh_generic_pl_srv_actual_state_info_t | | |
|---|---|---|
| Generic Power Actual state determines the linear percentage of the maximum power level of an element, representing a range from 0 percent through 100 percent | | |
| Data Fields | | |
| uint16_t | power_actual | Generic Power Actual |
| uint8_t | tid | |
| uint16_t | power_target | Generic Power Target - Used only in response path |
| uint8_t | transition_time | Transition Time - Used in request path Used as Remaining Time in response path |
| uint8_t | delay | |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_generic_pl_srv_last_state_info_t

| struct rm_mesh_generic_pl_srv_last_state_info_t | | |
|---|---|---|
| Generic Power Last state is a 16-bit value representing a percentage ranging from (1/65535) percent to 100 percent | | |
| Data Fields | | |
| uint16_t | power_last | Generic Power Last |

### ◆ rm_mesh_generic_pl_srv_default_state_info_t

| struct rm_mesh_generic_pl_srv_default_state_info_t | | |
|---|---|---|
| Generic Power Default state is a 16-bit value ranging from 0 through 65535 | | |
| Data Fields | | |
| uint16_t | power_default | Generic Power Default |

### ◆ rm_mesh_generic_pl_srv_range_state_info_t

| struct rm_mesh_generic_pl_srv_range_state_info_t |
|---|

| Generic Power Range state determines the minimum and maximum power levels of an element relative to the maximum power level an element can output | | |
|---|---|---|
| Data Fields | | |
| uint16_t | power_range_min | Generic Power Range - Minimum |
| uint16_t | power_range_max | Generic Power Range - Maximum |
| uint8_t | status | Status - Used only in the response path |

### ◆ rm_mesh_generic_pl_srv_state_info_t

| struct rm_mesh_generic_pl_srv_state_info_t | | |
|---|---|---|
| Generic Power Level state is a composite state that includes a Generic Power Actual state, a Generic Power Last state, a Generic Power Default state, and a Generic Power Range state | | |
| Data Fields | | |
| rm_mesh_generic_pl_srv_actual_state_info_t | generic_power_actual | Generic Power Actual |
| rm_mesh_generic_pl_srv_last_state_info_t | generic_power_last | Generic Power Last |
| rm_mesh_generic_pl_srv_default_state_info_t | generic_power_default | Generic Power Default |
| rm_mesh_generic_pl_srv_range_state_info_t | generic_power_range | Generic Power Range |

### ◆ rm_mesh_generic_pl_srv_instance_ctrl_t

| struct rm_mesh_generic_pl_srv_instance_ctrl_t |
|---|
| BLE mesh generic pl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_PL_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_PL_SRV_Open()

fsp_err_t RM_MESH_GENERIC_PL_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_Power_Level Server model and to initialize Generic_Power_Level_Setup Server model. This is to initialize Generic_Power_Level Server model and to register with Access layer. And this is to initialize Generic_Power_Level_Setup Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_PL_SRV_Open(&g_mesh_generic_pl_srv0_ctrl,
&g_mesh_generic_pl_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_PL_SRV_Close()

fsp_err_t RM_MESH_GENERIC_PL_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Power_Level Server model. This is to terminate Generic_Power_Level Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_PL_SRV_Close(&g_mesh_generic_pl_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_PL_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_PL_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_PL_SRV_StateUpdate(&g_mesh_generic_pl_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PL_SRV_SetupServerStateUpdate()

fsp_err_t RM_MESH_GENERIC_PL_SRV_SetupServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_pl_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Power On Off Client (rm_mesh_generic_poo_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_OnpowerupGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_OnpowerupSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_POO_CLT_OnpowerupSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Power On Off Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_poo_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_poo_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power On Off Client (rm_mesh_generic_poo_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power On Off Client (rm_mesh_generic_poo_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power on off client ISR occurs |

| | | | |
|---|---|---|---|
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_poo_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_poo_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_poo_clt_instance_ctrl_t

struct rm_mesh_generic_poo_clt_instance_ctrl_t

BLE mesh generic poo instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_POO_CLT_Open() is called.

## Function Documentation

### ◆ RM_MESH_GENERIC_POO_CLT_Open()

fsp_err_t RM_MESH_GENERIC_POO_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Power_Onoff Client middleware. This is to initialize Generic_Power_Onoff Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_POO_CLT_Open(&g_mesh_generic_poo_clt0_ctrl,
&g_mesh_generic_poo_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_POO_CLT_Close()

fsp_err_t RM_MESH_GENERIC_POO_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Power_Onoff Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_POO_CLT_Close(&g_mesh_generic_poo_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_POO_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_POO_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Power_Onoff client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic power on off client model. */
   err = RM_MESH_GENERIC_POO_CLT_GetModelHandle(&g_mesh_generic_poo_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_POO_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_POO_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_GENERIC_POO_CLT_SendReliablePdu(&g_mesh_generic_poo_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_POO_CLT_OnpowerupGet()

fsp_err_t RM_MESH_GENERIC_POO_CLT_OnpowerupGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic OnPowerUp Get is an acknowledged message used to get the Generic OnPowerUp state of an element. The response to the Generic OnPowerUp Get message is a Generic OnPowerUp Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_poo_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_POO_CLT_OnpowerupSet()

fsp_err_t RM_MESH_GENERIC_POO_CLT_OnpowerupSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic OnPowerUp Set is an acknowledged message used to set the Generic OnPowerUp state of an element. The response to the Generic OnPowerUp Set message is a Generic OnPowerUp Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_poo_clt control block. |
| [in] | p_parameter | Pointer to Generic OnPowerUp Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_POO_CLT_OnpowerupSetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_POO_CLT_OnpowerupSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic OnPowerUp Set Unacknowledged is an unacknowledged message used to set the Generic OnPowerUp state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_poo_clt control block. |
| [in] | p_parameter | Pointer to Generic OnPowerUp Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Power On Off Server (rm_mesh_generic_poo_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_POO_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_POO_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_POO_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_GENERIC_POO_SRV_SetupServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic Power On Off Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_poo_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_poo_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power On Off Server (rm_mesh_generic_poo_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Power On Off Server (rm_mesh_generic_poo_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power on off server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic power on off server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_poo_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_poo_srv_on_power_up_state_info_t |
| struct | rm_mesh_generic_poo_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_generic_poo_srv_on_power_up_state_info_t

| struct rm_mesh_generic_poo_srv_on_power_up_state_info_t | | |
|---|---|---|
| Generic OnPowerUp state is an enumeration representing the behavior of an element when powered up | | |
| Data Fields | | |
| uint8_t | onpowerup | Generic OnPowerUp |

### ◆ rm_mesh_generic_poo_srv_instance_ctrl_t

| struct rm_mesh_generic_poo_srv_instance_ctrl_t |
|---|
| BLE mesh generic poo instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_POO_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_POO_SRV_Open()

| fsp_err_t RM_MESH_GENERIC_POO_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* ) |
|---|
| API to initialize Generic_Power_Onoff Server model and to initialize Generic_Power_Onoff_Setup Server model. This is to initialize Generic_Power_Onoff Server model and to register with Access layer. And this is to initialize Generic_Power_Onoff_Setup Server model and to register with Access layer.<br><br>Implements rm_ble_mesh_model_server_api_t::open.<br><br>Example:<br>`/* Open the module. */`<br>`err = RM_MESH_GENERIC_POO_SRV_Open(&g_mesh_generic_poo_srv0_ctrl, &g_mesh_generic_poo_srv0_cfg);`<br><br>**Return values**<br>FSP_SUCCESS — Model opened successfully.<br>FSP_ERR_ASSERTION — Pointer to control block or configuration structure is NULL.<br>FSP_ERR_ALREADY_OPEN — Model is already open.<br>FSP_ERR_NOT_FOUND — The number of models has exceeded the limit.<br>FSP_ERR_ABORTED — Model initialization is failed. |

### ◆ RM_MESH_GENERIC_POO_SRV_Close()

fsp_err_t RM_MESH_GENERIC_POO_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_Power_Onoff Server model. This is to terminate Generic_Power_Onoff Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_POO_SRV_Close(&g_mesh_generic_poo_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_POO_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_POO_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_GENERIC_POO_SRV_StateUpdate(&g_mesh_generic_poo_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_POO_SRV_SetupServerStateUpdate()

fsp_err_t RM_MESH_GENERIC_POO_SRV_SetupServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_poo_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic Property Client (rm_mesh_generic_prop_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_UserPropertiesGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_UserPropertyGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_UserPropertySet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_UserPropertySetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_AdminPropertiesGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_AdminPropertyGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_AdminPropertySet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_AdminPropertySetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertiesGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertyGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySetUnacknowle |

| | |
|---|---|
| | dged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_GENERIC_PROP_CLT_ClientPropertiesGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Generic Property Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_prop_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_prop_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic Property Client (rm_mesh_generic_prop_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic Property Client (rm_mesh_generic_prop_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic property client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_prop_clt0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_generic_prop_clt_instance_ctrl_t |

**Data Structure Documentation**

◆ **rm_mesh_generic_prop_clt_instance_ctrl_t**

| struct rm_mesh_generic_prop_clt_instance_ctrl_t |
|---|
| |

BLE mesh generic prop instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_PROP_CLT_Open() is called.

### Function Documentation

#### ◆ RM_MESH_GENERIC_PROP_CLT_Open()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Generic_Property Client middleware. This is to initialize Generic_Property Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_PROP_CLT_Open(&g_mesh_generic_prop_clt0_ctrl,
&g_mesh_generic_prop_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

#### ◆ RM_MESH_GENERIC_PROP_CLT_Close()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Generic_Property Client middleware. Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_PROP_CLT_Close(&g_mesh_generic_prop_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_PROP_CLT_GetModelHandle()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Generic_Property client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of generic property client model. */
   err = RM_MESH_GENERIC_PROP_CLT_GetModelHandle(&g_mesh_generic_prop_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_GENERIC_PROP_CLT_SendReliablePdu()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_GENERIC_PROP_CLT_SendReliablePdu(&g_mesh_generic_prop_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_UserPropertiesGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_UserPropertiesGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic User Properties Get is an acknowledged message used to get the list of Generic User Property states of an element. The response to the Generic User Properties Get message is a Generic User Properties Status message. The message has no parameters.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_UserPropertyGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_UserPropertyGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic User Property Get is an acknowledged message used to get the Generic User Property state of an element. The response to the Generic User Property Get message is a Generic User Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic User Property Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_UserPropertySet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_UserPropertySet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic User Property Set is an acknowledged message used to set the Generic User Property state of an element. The response to the Generic User Property Set message is a Generic User Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic User Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_UserPropertySetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_UserPropertySetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic User Property Set Unacknowledged is an unacknowledged message used to set the Generic User Property state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic User Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_AdminPropertiesGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_AdminPropertiesGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Admin Properties Get is an acknowledged message used to get the list of Generic Admin Property states of an element. The response to the Generic Admin Properties Get message is a Generic Admin Properties Status message. The message has no parameters.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_AdminPropertyGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_AdminPropertyGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Admin Property Get is an acknowledged message used to get the Generic Admin Property state of an element. The response to the Generic Admin Property Get message is a Generic Admin Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Admin Property Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_AdminPropertySet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_AdminPropertySet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Admin Property Set is an acknowledged message used to set the Generic Admin Property state of an element. The response to the Generic Admin Property Set message is a Generic Admin Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Admin Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_AdminPropertySetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_AdminPropertySetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Admin Property Set Unacknowledged is an unacknowledged message used to set the Generic Admin Property state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Admin Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertiesGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertiesGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Generic Manufacturer Properties Get is an acknowledged message used to get the list of Generic Manufacturer Property states of an element. The response to the Generic Manufacturer Properties Get message is a Generic Manufacturer Properties Status message. The message has no parameters.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertyGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertyGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Manufacturer Property Get is an acknowledged message used to get the Generic Manufacturer Property state of an element. The response to the Generic Manufacturer Property Get message is a Generic Manufacturer Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Manufacturer Property Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Manufacturer Property Set is an acknowledged message used to set the Generic Manufacturer Property User Access state of an element. The response to the Generic Manufacturer Property Set message is a Generic Manufacturer Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Manufacturer Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySetUnacknowledged()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_ManufacturerPropertySetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Generic Manufacturer Property Set Unacknowledged is an unacknowledged message used to set the Generic Manufacturer Property User Access state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Manufacturer Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_GENERIC_PROP_CLT_ClientPropertiesGet()

fsp_err_t RM_MESH_GENERIC_PROP_CLT_ClientPropertiesGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Generic Client Properties Get is an acknowledged message used to get the list of Generic Client Property states of an element. The response to the Generic Client Properties Get message is a Generic Client Properties Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_generic_prop_clt control block. |
| [in] | p_parameter | Pointer to Generic Client Properties Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Generic User Property Server (rm_mesh_generic_user_prop_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_USER_PROP_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_GENERIC_USER_PROP_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_GENERIC_USER_PROP_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Generic User Property Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_generic_user_prop_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_generic_user_prop_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Generic User Property Server (rm_mesh_generic_user_prop_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Generic User Property Server (rm_mesh_generic_user_prop_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic user property server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh generic user property server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_generic_user_prop_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_generic_user_prop_srv_info_t |
| struct | rm_mesh_generic_user_prop_srv_instance_ctrl_t |

### Data Structure Documentation

### ◆ rm_mesh_generic_user_prop_srv_info_t

| struct rm_mesh_generic_user_prop_srv_info_t | | |
|---|---|---|
| Generic User Property is a state representing a device property of an element | | |
| Data Fields | | |
| uint16_t | property_id | User Property ID field is a 2-octet Assigned Number value referencing a device property |
| uint8_t | user_access | User Access field is an enumeration indicating whether the device property can be read or written as a Generic User Property |
| uint8_t * | property_value | User Property Value field is a conditional field |
| uint16_t | property_value_len | |

### ◆ rm_mesh_generic_user_prop_srv_instance_ctrl_t

| struct rm_mesh_generic_user_prop_srv_instance_ctrl_t |
|---|
| BLE mesh generic user prop instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_GENERIC_USER_PROP_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_GENERIC_USER_PROP_SRV_Open()

fsp_err_t RM_MESH_GENERIC_USER_PROP_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Generic_User_Property Server model. This is to initialize Generic_User_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_GENERIC_USER_PROP_SRV_Open(&g_mesh_generic_user_prop_srv0_ctrl,
&g_mesh_generic_user_prop_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_GENERIC_USER_PROP_SRV_Close()

fsp_err_t RM_MESH_GENERIC_USER_PROP_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Generic_User_Property Server model. This is to terminate Generic_User_Property Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_GENERIC_USER_PROP_SRV_Close(&g_mesh_generic_user_prop_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_GENERIC_USER_PROP_SRV_StateUpdate()

fsp_err_t RM_MESH_GENERIC_USER_PROP_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err =
RM_MESH_GENERIC_USER_PROP_SRV_StateUpdate(&g_mesh_generic_user_prop_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Health Client (rm_mesh_health_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_HEALTH_CLIENT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_HEALTH_CLIENT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_FaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_FaultClearUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_FaultClear (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_FaultTest (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_FaultTestUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_PeriodGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_PeriodSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_PeriodSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_AttentionGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_AttentionSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_HEALTH_CLIENT_AttentionSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Health Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_health_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_health_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Health Client (rm_mesh_health_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Health Client (rm_mesh_health_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh health client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_health_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_health_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_health_clt_instance_ctrl_t

| struct rm_mesh_health_clt_instance_ctrl_t |
|---|
| BLE mesh health client instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_HEALTH_CLIENT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_HEALTH_CLIENT_Open()

fsp_err_t RM_MESH_HEALTH_CLIENT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Health Client middleware. This is to initialize Health Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_HEALTH_CLIENT_Open(&g_mesh_health_clt0_ctrl,
&g_mesh_health_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_Close()

fsp_err_t RM_MESH_HEALTH_CLIENT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Health Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_HEALTH_CLIENT_Close(&g_mesh_health_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_HEALTH_CLIENT_GetModelHandle()

fsp_err_t RM_MESH_HEALTH_CLIENT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Health client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of health client model. */
   err = RM_MESH_HEALTH_CLIENT_GetModelHandle(&g_mesh_health_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_HEALTH_CLIENT_SendReliablePdu()

fsp_err_t RM_MESH_HEALTH_CLIENT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to set the handle of Health client model.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_HEALTH_CLIENT_SendReliablePdu(&g_mesh_health_clt0_ctrl, req_opcode,
p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_FaultGet()

fsp_err_t RM_MESH_HEALTH_CLIENT_FaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Fault Get is an acknowledged message used to get the current Registered Fault state identified by Company ID of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_fault_get_clear_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_FaultClearUnacknowledged()

fsp_err_t RM_MESH_HEALTH_CLIENT_FaultClearUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Fault Clear Unacknowledged is an unacknowledged message used to clear the current Registered Fault state identified by Company ID of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_fault_get_clear_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_FaultClear()

fsp_err_t RM_MESH_HEALTH_CLIENT_FaultClear ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Fault Clear is an acknowledged message used to clear the current Registered Fault state identified by Company ID of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_fault_get_clear_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_FaultTest()

fsp_err_t RM_MESH_HEALTH_CLIENT_FaultTest ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Fault Test is an acknowledged message used to invoke a self-test procedure of an element. The procedure is implementation specific and may result in changing the Health Fault state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_fault_test_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_FaultTestUnacknowledged()

fsp_err_t RM_MESH_HEALTH_CLIENT_FaultTestUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Fault Test Unacknowledged is an unacknowledged message used to invoke a self-test procedure of an element. The procedure is implementation specific and may result in changing the Health Fault state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_fault_test_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_PeriodGet()

fsp_err_t RM_MESH_HEALTH_CLIENT_PeriodGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Health Period Get is an acknowledged message used to get the current Health Period state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_PeriodSetUnacknowledged()

fsp_err_t RM_MESH_HEALTH_CLIENT_PeriodSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Period Set Unacknowledged is an unacknowledged message used to set the current Health Period state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_period_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_PeriodSet()

fsp_err_t RM_MESH_HEALTH_CLIENT_PeriodSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Period Set is an acknowledged message used to set the current Health Period state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_period_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_AttentionGet()

fsp_err_t RM_MESH_HEALTH_CLIENT_AttentionGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Health Attention Get is an acknowledged message used to get the current Attention Timer state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_AttentionSet()

fsp_err_t RM_MESH_HEALTH_CLIENT_AttentionSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Attention Set is an acknowledged message used to set the Attention Timer state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_attention_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_HEALTH_CLIENT_AttentionSetUnacknowledged()

fsp_err_t RM_MESH_HEALTH_CLIENT_AttentionSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Health Attention Set Unacknowledged is an unacknowledged message used to set the Attention Timer state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_clt control block. |
| [in] | p_parameter | Pointer to the structure populated as in rm_mesh_health_clt_attention_info_t. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Health Server (rm_mesh_health_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_HEALTH_SERVER_Open (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, rm_ble_mesh_health_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_HEALTH_SERVER_Close (rm_ble_mesh_health_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_HEALTH_SERVER_ReportFault (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, const rm_ble_mesh_access_model_handle_t *const model_handle, uint8_t test_id, uint16_t company_id, uint8_t fault_code) |
| fsp_err_t | RM_MESH_HEALTH_SERVER_PublishCurrentStatus (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, uint8_t *status, uint16_t length) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Health Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_health_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_health_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Health Server (rm_mesh_health_srv)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Health Server (rm_mesh_health_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh health server ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_health_srv0 | Module name. |
| Company ID | Invalid Company Id | 0 | Select company id. |
| Number of Self Tests | Invalid Number of Self Tests | 0 | Select number of self tests. |
| Self Tests | Name Must Be a Valid C Symbol | NULL | Self test |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_health_server_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_health_server_instance_ctrl_t

| struct rm_mesh_health_server_instance_ctrl_t |
|---|
| BLE mesh health server instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_HEALTH_SERVER_Open() is called. |

## Function Documentation

### ◆ RM_MESH_HEALTH_SERVER_Open()

fsp_err_t RM_MESH_HEALTH_SERVER_Open ( rm_ble_mesh_health_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_health_server_cfg_t const *const *p_cfg* )

API to initialize Health Server model. This is to initialize Health Server model and to register with Access layer.

Implements rm_ble_mesh_health_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_HEALTH_SERVER_Open(&g_mesh_health_srv0_ctrl,
&g_mesh_health_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_HEALTH_SERVER_Close()

fsp_err_t RM_MESH_HEALTH_SERVER_Close ( rm_ble_mesh_health_server_ctrl_t *const *p_ctrl*)

API to terminate Health Server model. This is to terminate Health Server model and to register with Access layer.

Implements rm_ble_mesh_health_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_HEALTH_SERVER_Close(&g_mesh_health_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_HEALTH_SERVER_ReportFault()

fsp_err_t RM_MESH_HEALTH_SERVER_ReportFault ( rm_ble_mesh_health_server_ctrl_t *const *p_ctrl*, const rm_ble_mesh_access_model_handle_t *const *model_handle*, uint8_t *test_id*, uint16_t *company_id*, uint8_t *fault_code* )

API to report self-test fault. This is to report fault observed during self-test procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_srv control block. |
| [in] | model_handle | Model Handle identifying the Health Server model instance. |
| [in] | test_id | Identifier of the self-test. |
| [in] | company_id | Company Identifier. |
| [in] | fault_code | Fault value indicating the error. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_MESH_HEALTH_SERVER_PublishCurrentStatus()

fsp_err_t RM_MESH_HEALTH_SERVER_PublishCurrentStatus ( rm_ble_mesh_health_server_ctrl_t *const *p_ctrl*, uint8_t * *status*, uint16_t *length* )

API to publish current status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_health_srv control block. |
| [in] | status | Current status. |
| [in] | length | Length of status. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Control Client (rm_mesh_light_ctl_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_GetModelHandle |

| | |
|---|---|
| | (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_DefaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_DefaultSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_DefaultSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureRangeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Control Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_ctl_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_light_ctl_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Control Client (rm_mesh_light_ctl_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Control Client (rm_mesh_light_ctl_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light control client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_ctl_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_ctl_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_light_ctl_clt_instance_ctrl_t

| struct rm_mesh_light_ctl_clt_instance_ctrl_t |
|---|
| BLE mesh light ctl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_CTL_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_LIGHT_CTL_CLT_Open()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Light_Ctl Client middleware. This is to initialize Light_Ctl Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_CTL_CLT_Open(&g_mesh_light_ctl_clt0_ctrl,
&g_mesh_light_ctl_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_Close()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Light_Ctl Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_CTL_CLT_Close(&g_mesh_light_ctl_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_CTL_CLT_GetModelHandle()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Light_Ctl client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of light ctl client model. */
   err = RM_MESH_LIGHT_CTL_CLT_GetModelHandle(&g_mesh_light_ctl_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_LIGHT_CTL_CLT_SendReliablePdu()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_LIGHT_CTL_CLT_SendReliablePdu(&g_mesh_light_ctl_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_Get()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light CTL Get is an acknowledged message used to get the Light CTL state of an element. The response to the Light CTL Get message is a Light CTL Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_Set()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light CTL Set is an acknowledged message used to set the Light CTL Lightness state, Light CTL Temperature state, and the Light CTL Delta UV state of an element. The response to the Light CTL Set message is a Light CTL Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light CTL Set Unacknowledged is an unacknowledged message used to set the Light CTL Lightness state, Light CTL Temperature state, and the Light CTL Delta UV state of an element

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureGet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light CTL Temperature Get is an acknowledged message used to get the Light CTL Temperature state of an element. The response to the Light CTL Temperature Get message is a Light CTL Temperature Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureSet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light CTL Temperature Set is an acknowledged message used to set the Light CTL Temperature state and the Light CTL Delta UV state of an element. The response to the Light CTL Temperature Set message is a Light CTL Temperature Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Temperature Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light CTL Temperature Set Unacknowledged is an unacknowledged message used to set the Light CTL Temperature state and the Light CTL Delta UV state of an element

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Temperature Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_DefaultGet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_DefaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light CTL Default Get is an acknowledged message used to get the Light CTL Temperature Default and Light CTL Delta UV Default states of an element. The response to the Light CTL Default Get message is a Light CTL Default Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_DefaultSet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_DefaultSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light CTL Default Set is an acknowledged message used to set the Light CTL Temperature Default state and the Light CTL Delta UV Default state of an element. The response to the Light CTL Set message is a Light CTL Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_DefaultSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_DefaultSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light CTL Default Set Unacknowledged is an unacknowledged message used to set the Light CTL Temperature Default state and the Light CTL Delta UV Default state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureRangeGet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureRangeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light CTL Temperature Range Get is an acknowledged message used to get the Light CTL Temperature Range state of an element. The response to the Light CTL Temperature Range Get message is a Light CTL Temperature Range Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSet()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light CTL Temperature Range Set Unacknowledged is an unacknowledged message used to set the Light CTL Temperature Range state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Temperature Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_CTL_CLT_TemperatureRangeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light CTL Temperature Range Set is an acknowledged message used to set the Light CTL Temperature Range state of an element. The response to the Light CTL Temperature Range Get message is a Light CTL Temperature Range Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_clt control block. |
| [in] | p_parameter | Pointer to Light CTL Temperature Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Control Server (rm_mesh_light_ctl_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_CTL_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_CTL_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_CTL_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_LIGHT_CTL_SRV_TemperatureServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Light Control Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_ctl_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_light_ctl_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Control Server (rm_mesh_light_ctl_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Control Server (rm_mesh_light_ctl_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light control server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light control server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_ctl_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_light_ctl_srv_info_t |

| | |
|---|---|
| struct | rm_mesh_light_ctl_srv_default_info_t |
| struct | rm_mesh_light_ctl_srv_temperature_info_t |
| struct | rm_mesh_light_ctl_srv_temperature_range_info_t |
| struct | rm_mesh_light_ctl_srv_extended_cfg_t |
| struct | rm_mesh_light_ctl_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_ctl_srv_info_t

| struct rm_mesh_light_ctl_srv_info_t | | |
|---|---|---|
| Light CTL state is a composite state that includes the Light CTL Lightness, the Light CTL Temperature and the Light CTL Delta UV states | | |
| Data Fields | | |
| uint16_t | ctl_lightness | Light CTL Lightness |
| uint16_t | target_ctl_lightness | Target Light CTL Lightness - Used in response path |
| uint16_t | ctl_temperature | Light CTL Temperature |
| uint16_t | target_ctl_temperature | Terget Light CTL Temperature - Used in response path |
| uint16_t | ctl_delta_uv | Light CTL Delta UV |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_light_ctl_srv_default_info_t

| struct rm_mesh_light_ctl_srv_default_info_t | | |
|---|---|---|
| Light CTL Default state is a composite state that includes the Light CTL Lightness, the Light CTL Temperature and the Light CTL Delta UV states | | |
| Data Fields | | |
| uint16_t | ctl_lightness | Light CTL Lightness |
| uint16_t | ctl_temperature | Light CTL Temperature |
| uint16_t | ctl_delta_uv | Light CTL Delta UV |

### ◆ rm_mesh_light_ctl_srv_temperature_info_t

| struct rm_mesh_light_ctl_srv_temperature_info_t | | |
|---|---|---|
| Light CTL Temperature state is a composite state that includes the Light CTL Temperature and the Light CTL Delta UV states | | |
| Data Fields | | |
| uint16_t | ctl_temperature | Light CTL Temperature |
| uint16_t | target_ctl_temperature | Target Light CTL Temperature - Used in response path |
| uint16_t | ctl_delta_uv | Light CTL Delta UV |
| uint16_t | target_ctl_delta_uv | Target Light CTL Delta UV - Used in response path |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |

### ◆ rm_mesh_light_ctl_srv_temperature_range_info_t

| struct rm_mesh_light_ctl_srv_temperature_range_info_t | | |
|---|---|---|
| Light CTL Temperature Range state determines the minimum and maximum color temperatures of tunable white light an element is capable of emitting | | |
| Data Fields | | |
| uint16_t | ctl_temperature_range_min | CTL Temperature Range Min |
| uint16_t | ctl_temperature_range_max | CTL Temperature Range Max |
| uint8_t | status | Status - Used in response path |

### ◆ rm_mesh_light_ctl_srv_extended_cfg_t

| struct rm_mesh_light_ctl_srv_extended_cfg_t | | |
|---|---|---|
| BLE mesh light ctl extension for BLE mesh light ctl. | | |
| Data Fields | | |
| rm_ble_mesh_access_instance_t const * | p_temperature_access_instance | Pointer to access instance for temperature model. |

### ◆ rm_mesh_light_ctl_srv_instance_ctrl_t

| struct rm_mesh_light_ctl_srv_instance_ctrl_t |
|---|
| BLE mesh light ctl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_CTL_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_CTL_SRV_Open()

fsp_err_t RM_MESH_LIGHT_CTL_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Light_Ctl Server model and to initialize Light_Ctl_Temperature Server model. This is to initialize Light_Ctl Server model and to register with Access layer. And this is to initialize Light_Ctl_Temperature Server model and to register with Access layer. Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_CTL_SRV_Open(&g_mesh_light_ctl_srv0_ctrl,
&g_mesh_light_ctl_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_CTL_SRV_Close()

fsp_err_t RM_MESH_LIGHT_CTL_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Light_Ctl Server model. This is to terminate Light_Ctl Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_CTL_SRV_Close(&g_mesh_light_ctl_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_CTL_SRV_StateUpdate()

fsp_err_t RM_MESH_LIGHT_CTL_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_LIGHT_CTL_SRV_StateUpdate(&g_mesh_light_ctl_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_CTL_SRV_TemperatureServerStateUpdate()

fsp_err_t RM_MESH_LIGHT_CTL_SRV_TemperatureServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_ctl_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Hsl Client (rm_mesh_light_hsl_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_TargetGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_DefaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_DefaultSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_DefaultSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_RangeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_RangeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_RangeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_HueGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_HueSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_HueSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const |

| | |
|---|---|
| | p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_SaturationGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_SaturationSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_HSL_CLT_SaturationSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Hsl Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_hsl_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_light_hsl_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light HSL Client (rm_mesh_light_hsl_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light HSL Client (rm_mesh_light_hsl_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light HSL client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_hsl_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_hsl_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_hsl_clt_instance_ctrl_t

| struct rm_mesh_light_hsl_clt_instance_ctrl_t |
|---|
| BLE mesh light hsl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_HSL_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_HSL_CLT_Open()

| fsp_err_t RM_MESH_LIGHT_HSL_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* ) |
|---|
| Open Light_Hsl Client middleware. This is to initialize Light_Hsl Client model and to register with Access layer.<br><br>Implements rm_ble_mesh_model_client_api_t::open.<br><br>Example:<br>`/* Open the module. */`<br>`err = RM_MESH_LIGHT_HSL_CLT_Open(&g_mesh_light_hsl_clt0_ctrl, &g_mesh_light_hsl_clt0_cfg);` |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_Close()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Light_Hsl Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_HSL_CLT_Close(&g_mesh_light_hsl_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_HSL_CLT_GetModelHandle()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Light_Hsl client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of light hsl client model. */
   err = RM_MESH_LIGHT_HSL_CLT_GetModelHandle(&g_mesh_light_hsl_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_LIGHT_HSL_CLT_SendReliablePdu()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_LIGHT_HSL_CLT_SendReliablePdu(&g_mesh_light_hsl_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_Get()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light HSL Get is an acknowledged message used to get the Light HSL Lightness, Light HSL Hue, and Light HSL Saturation states of an element. The response to the Light HSL Get message is a Light HSL Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_Set()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Set Unacknowledged is an unacknowledged message used to set the Light HSL Lightness state, Light HSL Hue state, and the Light HSL Saturation state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Set is an acknowledged message used to set the Light HSL Lightness state, Light HSL Hue state, and the Light HSL Saturation state of an element. The response to the Light HSL Set message is a Light HSL Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_TargetGet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_TargetGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light HSL Target Get is an acknowledged message used to get the target Light HSL Lightness, Light HSL Hue, and Light HSL Saturation states of an element. The response to the Light HSL Target Get message is a Light HSL Target Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_DefaultGet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_DefaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light HSL Default Get is an acknowledged message used to get the Light Lightness Default, the Light HSL Hue Default, and Light HSL Saturation Default states of an element. The response to the Light HSL Default Get message is a Light HSL Default Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_DefaultSet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_DefaultSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light HSL Default Set is an acknowledged message used to set the Light Lightness Default, the Light HSL Hue Default, and Light HSL Saturation Default states of an element. The response to the Light HSL Default Set message is a Light HSL Default Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_DefaultSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_DefaultSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light HSL Default Set Unacknowledged is an unacknowledged message used to set the Light Lightness Default, the Light HSL Hue Default, and Light HSL Saturation Default states of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_RangeGet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_RangeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light HSL Range Get is an acknowledged message used to get the Light HSL Hue Range and Light HSL Saturation Range states of an element. The response to the Light HSL Range Get message is a Light HSL Range Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_RangeSet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_RangeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light HSL Range Set is an acknowledged message used to set the Light HSL Hue Range and Light HSL Saturation Range states of an element. The response to the Light HSL Range Set message is a Light HSL Range Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_RangeSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_RangeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light HSL Range Set Unacknowledged is an unacknowledged message used to set the Light HSL Hue Range and Light HSL Saturation Range states of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_HueGet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_HueGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light HSL Hue Get is an acknowledged message used to get the Light HSL Hue state of an element. The response to the Light HSL Hue Get message is a Light HSL Hue Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_HueSet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_HueSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Hue Set is an acknowledged message used to set the target Light HSL Hue state of an element. The response to the Light HSL Hue Set message is a Light HSL Hue Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Hue Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_HueSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_HueSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Hue Set Unacknowledged is an unacknowledged message used to set the target Light HSL Hue state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Hue Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_SaturationGet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_SaturationGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light HSL Saturation Get is an acknowledged message used to get the Light HSL Saturation state of an element. The response to the Light HSL Saturation Get message is a Light HSL Saturation Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_SaturationSet()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_SaturationSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Saturation Set is an acknowledged message used to set the target Light HSL Saturation state of an element. The response to the Light HSL Saturation Set message is a Light HSL Saturation Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Saturation Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_CLT_SaturationSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_HSL_CLT_SaturationSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light HSL Saturation Set Unacknowledged is an unacknowledged message used to set the target Light HSL Saturation state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Saturation Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Hsl Server (rm_mesh_light_hsl_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_HSL_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_HSL_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_HSL_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_LIGHT_HSL_SRV_HueServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_LIGHT_HSL_SRV_SaturationServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Light Hsl Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_hsl_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_light_hsl_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light HSL Server (rm_mesh_light_hsl_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light HSL Server (rm_mesh_light_hsl_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light HSL server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light HSL server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_hsl_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_light_hsl_srv_info_t |

| | |
|---|---|
| struct | rm_mesh_light_hsl_srv_target_info_t |
| struct | rm_mesh_light_hsl_srv_default_info_t |
| struct | rm_mesh_light_hsl_srv_hue_info_t |
| struct | rm_mesh_light_hsl_srv_saturation_info_t |
| struct | rm_mesh_light_hsl_srv_range_info_t |
| struct | rm_mesh_light_hsl_srv_extended_cfg_t |
| struct | rm_mesh_light_hsl_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_hsl_srv_info_t

struct rm_mesh_light_hsl_srv_info_t

Light HSL state is a composite state that includes the Light HSL Lighness, the Light HSL Hue and the Light HSL Saturation states

Data Fields

| | | |
|---|---|---|
| uint16_t | hsl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | target_hsl_lightness | Target Perceived lightness - used in the response path |
| uint16_t | hsl_hue | The 16-bit value representing the hue |
| uint16_t | target_hsl_hue | Target hue - used in the response path |
| uint16_t | hsl_saturation | The saturation of a color light |
| uint16_t | target_hsl_saturation | Target saturation - used in the response path |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_light_hsl_srv_target_info_t

struct rm_mesh_light_hsl_srv_target_info_t

| Light HSL Target state is a composite state that includes the Light HSL Lighness, the Light HSL Hue and the Light HSL Saturation states | | |
|---|---|---|
| Data Fields | | |
| uint16_t | hsl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | hsl_hue | The 16-bit value representing the hue |
| uint16_t | hsl_saturation | The saturation of a color light |

### ◆ rm_mesh_light_hsl_srv_default_info_t

| struct rm_mesh_light_hsl_srv_default_info_t | | |
|---|---|---|
| Light HSL Default state is a composite state that includes the Light HSL Lighness, the Light HSL Hue and the Light HSL Saturation states | | |
| Data Fields | | |
| uint16_t | hsl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | hsl_hue | The 16-bit value representing the hue |
| uint16_t | hsl_saturation | The saturation of a color light |

### ◆ rm_mesh_light_hsl_srv_hue_info_t

| struct rm_mesh_light_hsl_srv_hue_info_t | | |
|---|---|---|
| Light HSL Hue | | |
| Data Fields | | |
| uint16_t | hsl_hue | The 16-bit value representing the hue |

### ◆ rm_mesh_light_hsl_srv_saturation_info_t

| struct rm_mesh_light_hsl_srv_saturation_info_t | | |
|---|---|---|
| Light HSL Saturation | | |
| Data Fields | | |
| uint16_t | hsl_saturation | The saturation of a color light |

### ◆ rm_mesh_light_hsl_srv_range_info_t

| struct rm_mesh_light_hsl_srv_range_info_t | | |
|---|---|---|
| Light HSL Range state is a composite state that includes Minimum and Maximum of the Light HSL Hue and the Light HSL Saturation states | | |
| Data Fields | | |
| uint16_t | hue_range_min | The value of the Hue Range Min field of the Light HSL Hue |

| | | |
|---|---|---|
| | | Range state |
| uint16_t | hue_range_max | The value of the Hue Range Max field of the Light HSL Hue Range state |
| uint16_t | saturation_range_min | The value of the Saturation Range Min field of the Light HSL Saturation Range state |
| uint16_t | saturation_range_max | The value of the Saturation Range Max field of the Light HSL Saturation Range state |
| uint8_t | status | Status - Used only in response path |

### ◆ rm_mesh_light_hsl_srv_extended_cfg_t

| struct rm_mesh_light_hsl_srv_extended_cfg_t | | |
|---|---|---|
| BLE mesh light hsl extension for BLE mesh light hsl. | | |
| Data Fields | | |
| rm_ble_mesh_access_instance_t const * | p_hue_access_instance | Pointer to access instance for hue model. |
| rm_ble_mesh_access_instance_t const * | p_saturation_access_instance | Pointer to access instance for saturation model. |

### ◆ rm_mesh_light_hsl_srv_instance_ctrl_t

| struct rm_mesh_light_hsl_srv_instance_ctrl_t |
|---|
| BLE mesh light hsl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_HSL_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_HSL_SRV_Open()

fsp_err_t RM_MESH_LIGHT_HSL_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Light_Hsl Server/Light_Hsl_Hue Server/Light_Hsl_Saturation Server model.

1. This is to initialize Light_Hsl Server model and to register with Access layer.
2. This is to initialize Light_Hsl_Hue Server model and to register with Access layer.
3. This is to initialize Light_Hsl_Saturation Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_HSL_SRV_Open(&g_mesh_light_hsl_srv0_ctrl,
&g_mesh_light_hsl_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_HSL_SRV_Close()

fsp_err_t RM_MESH_LIGHT_HSL_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Light_Hsl Server model. This is to terminate Light_Hsl Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_HSL_SRV_Close(&g_mesh_light_hsl_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_HSL_SRV_StateUpdate()

fsp_err_t RM_MESH_LIGHT_HSL_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_LIGHT_HSL_SRV_StateUpdate(&g_mesh_light_hsl_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_SRV_HueServerStateUpdate()

fsp_err_t RM_MESH_LIGHT_HSL_SRV_HueServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_HSL_SRV_SaturationServerStateUpdate()

fsp_err_t RM_MESH_LIGHT_HSL_SRV_SaturationServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_hsl_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Lightness Client (rm_mesh_light_lightness_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_LinearGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_LastGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_RangeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Lightness Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_lightness_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_light_lightness_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Client (rm_mesh_light_lightness_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Client (rm_mesh_light_lightness_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_lightness_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_lightness_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_light_lightness_clt_instance_ctrl_t

| struct rm_mesh_light_lightness_clt_instance_ctrl_t |
|---|
| BLE mesh light lightness instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_LIGHTNESS_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_Open()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Light_Lightness Client middleware. This is to initialize Light_Lightness Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_LIGHTNESS_CLT_Open(&g_mesh_light_lightness_clt0_ctrl,
&g_mesh_light_lightness_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_Close()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Light_Lightness Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_LIGHTNESS_CLT_Close(&g_mesh_light_lightness_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_GetModelHandle()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Light_Lightness client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of light lightness client model. */
   err =
RM_MESH_LIGHT_LIGHTNESS_CLT_GetModelHandle(&g_mesh_light_lightness_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_SendReliablePdu()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err =
RM_MESH_LIGHT_LIGHTNESS_CLT_SendReliablePdu(&g_mesh_light_lightness_clt0_ctrl,
                                            req_opcode,
                                            p_parameter,
                                            rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_Get()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light Lightness Get is an acknowledged message used to get the Light Lightness Actual state of an element. The response to the Light Lightness Get message is a Light Lightn

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_Set()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Set is an acknowledged message used to set the Light Lightness Actual state of an element. The response to the Light Lightness Set message is a Light Lightness Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Set Unacknowledged is an unacknowledged message used to set the Light Lightness Actual state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_LinearGet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_LinearGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light Lightness Linear Get is an acknowledged message used to get the Light Lightness Linear state of an element. The response to the Light Lightness Linear Get message is a Light Lightness Linear Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Linear Set is an acknowledged message used to set the Light Lightness Linear state of an element. The response to the Light Lightness Linear Set message is a Light Lightness Linear Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Linear Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_LinearSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Linear Set Unacknowledged is an unacknowledged message used to set the Light Lightness Linear state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Linear Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_LastGet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_LastGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light Lightness Last Get is an acknowledged message used to get the Light Lightness Last state of an element. The response to the Light Lightness Last Get message is a Light Lightness Last Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultGet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light Lightness Default Get is an acknowledged message used to get the Light Lightness Default state of an element. The response to the Light Lightness Default Get message is a Light Lightness Default Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Default Set is an acknowledged message used to set the Light Lightness Default state of an element. The response to the Light Lightness Default Set message is a Light Lightness Default Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_DefaultSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light Lightness Default Set Unacknowledged is an unacknowledged message used to set the Light Lightness Default state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_RangeGet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_RangeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light Lightness Range Get is an acknowledged message used to get the Light Lightness Range state of an element. The response to the Light Lightness Range Get message is a Light Lightness Range Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSet()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light Lightness Range Set is an acknowledged message used to set the Light Lightness Range state of an element. The response to the Light Lightness Range Get message is a Light Lightness Range Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_CLT_RangeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light Lightness Range Set Unacknowledged is an unacknowledged message used to set the Light Lightness Range state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_clt control block. |
| [in] | p_parameter | Pointer to Light Lightness Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Lightness Server (rm_mesh_light_lightness_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_LIGHT_LIGHTNESS_SRV_SetupServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Lightness Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_lightness_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_light_lightness_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Server (rm_mesh_light_lightness_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Server (rm_mesh_light_lightness_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_lightness_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_lightness_srv_linear_info_t |
| struct | rm_mesh_light_lightness_srv_actual_info_t |

| | |
|---|---|
| struct | rm_mesh_light_lightness_srv_last_info_t |
| struct | rm_mesh_light_lightness_srv_default_info_t |
| struct | rm_mesh_light_lightness_srv_range_info_t |
| struct | rm_mesh_light_lightness_srv_info_t |
| struct | rm_mesh_light_lightness_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_lightness_srv_linear_info_t

| struct rm_mesh_light_lightness_srv_linear_info_t | | |
|---|---|---|
| Light Lightness Linear state represents the lightness of a light on a linear scale | | |
| Data Fields | | |
| uint16_t | lightness_linear | Light Lightness Linear |
| uint16_t | lightness_target | Light Lightness Target - Used in response path. |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |

### ◆ rm_mesh_light_lightness_srv_actual_info_t

| struct rm_mesh_light_lightness_srv_actual_info_t | | |
|---|---|---|
| Light Lightness Actual state represents the lightness of a light on a perceptually uniform lightness scale | | |
| Data Fields | | |
| uint16_t | lightness_actual | Light Lightness Actual |
| uint16_t | lightness_target | Light Lightness Target - Used in response path. |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_light_lightness_srv_last_info_t

| struct rm_mesh_light_lightness_srv_last_info_t | | |
|---|---|---|
| Light Lightness Last state represents the lightness of a light on a perceptually uniform lightness scale | | |
| Data Fields | | |
| uint16_t | lightness_last | Light Lightness Last |

### ◆ rm_mesh_light_lightness_srv_default_info_t

| struct rm_mesh_light_lightness_srv_default_info_t | | |
|---|---|---|
| Light Lightness Default state is a value ranging from 0x0000 to 0xFFFF, representing a default lightness level for the Light Lightness Actual state | | |
| Data Fields | | |
| uint16_t | lightness_default | Light Lightness Default |

### ◆ rm_mesh_light_lightness_srv_range_info_t

| struct rm_mesh_light_lightness_srv_range_info_t | | |
|---|---|---|
| Light Lightness Range state determines the minimum and maximum lightness of an element | | |
| Data Fields | | |
| uint16_t | lightness_range_min | Light Lightness Range Min |
| uint16_t | lightness_range_max | Light Lightness Range Max |

### ◆ rm_mesh_light_lightness_srv_info_t

| struct rm_mesh_light_lightness_srv_info_t | | |
|---|---|---|
| Light Lightness state is a composite state that includes the Light Lightness Linear, the Light Lightness Actual, the Light Lightness Last, and the Light Lightness Default states | | |
| Data Fields | | |
| rm_mesh_light_lightness_srv_linear_info_t | light_lightness_linear | Light Lightness Linear state represents the lightness of a light on a linear scale |
| rm_mesh_light_lightness_srv_actual_info_t | light_lightness_actual | Light Lightness Actual state represents the lightness of a light on a perceptually uniform lightness scale |
| rm_mesh_light_lightness_srv_last_info_t | light_lightness_last | Light Lightness Last state represents the lightness of a light on a perceptually uniform lightness scale |
| rm_mesh_light_lightness_srv_default_info_t | light_lightness_default | Light Lightness Default state is a value ranging from 0x0000 to 0xFFFF, representing a default lightness level for the Light Lightness Actual state |

| | | |
|---|---|---|
| rm_mesh_light_lightness_srv_range_info_t | light_lightness_range | Light Lightness Range state. |
| uint8_t | range_status | Status field used only for the Range Status |

### ◆ rm_mesh_light_lightness_srv_instance_ctrl_t

| struct rm_mesh_light_lightness_srv_instance_ctrl_t |
|---|
| BLE mesh light lightness instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_LIGHTNESS_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_LIGHTNESS_SRV_Open()

| fsp_err_t RM_MESH_LIGHT_LIGHTNESS_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* ) |
|---|

API to initialize Light_Lightness Server model and to initialize Light_Lightness_Setup Server model. This is to initialize Light_Lightness Server model and to register with Access layer. And this is to initialize Light_Lightness_Setup Server model and to register with Access layer. Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_LIGHTNESS_SRV_Open(&g_mesh_light_lightness_srv0_ctrl,
&g_mesh_light_lightness_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_SRV_Close()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Light_Lightness Server model. This is to terminate Light_Lightness Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_LIGHTNESS_SRV_Close(&g_mesh_light_lightness_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_SRV_StateUpdate()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_LIGHT_LIGHTNESS_SRV_StateUpdate(&g_mesh_light_lightness_srv0_ctrl,
&state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LIGHTNESS_SRV_SetupServerStateUpdate()

fsp_err_t RM_MESH_LIGHT_LIGHTNESS_SRV_SetupServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lightness_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Location Client (rm_mesh_light_lc_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_ModeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_ModeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_ModeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_OmGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_OmSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_OmSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_LightOnOffGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_LightOnOffSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_LightOnOffSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_PropertyGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_PropertySet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_LC_CLT_PropertySetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Location Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_lc_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_light_lc_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Controller Client (rm_mesh_light_lc_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Controller Client (rm_mesh_light_lc_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness controller client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_lc_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_lc_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_light_lc_clt_instance_ctrl_t

| struct rm_mesh_light_lc_clt_instance_ctrl_t |
|---|
| BLE mesh light lc instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_LC_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_LIGHT_LC_CLT_Open()

fsp_err_t RM_MESH_LIGHT_LC_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Light_Lc Client middleware. This is to initialize Light_Lc Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_LC_CLT_Open(&g_mesh_light_lc_clt0_ctrl,
&g_mesh_light_lc_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_Close()

fsp_err_t RM_MESH_LIGHT_LC_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Light_Lc Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_LC_CLT_Close(&g_mesh_light_lc_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LC_CLT_GetModelHandle()

fsp_err_t RM_MESH_LIGHT_LC_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Light_Lc client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of light lc client model. */
   err = RM_MESH_LIGHT_LC_CLT_GetModelHandle(&g_mesh_light_lc_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_LIGHT_LC_CLT_SendReliablePdu()

fsp_err_t RM_MESH_LIGHT_LC_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_LIGHT_LC_CLT_SendReliablePdu(&g_mesh_light_lc_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_ModeGet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_ModeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light LC Mode Get is an acknowledged message used to get the Light LC Mode state of an element. The response to the Light LC Mode Get message is a Light LC Mode Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_ModeSet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_ModeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Mode Set is an acknowledged message used to set the Light LC Mode state of an element. The response to the Light LC Mode Set message is a Light LC Mode Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Mode Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_ModeSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LC_CLT_ModeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Mode Set Unacknowledged is an unacknowledged message used to set the Light LC Mode state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Mode Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_OmGet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_OmGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light LC OM Get is an acknowledged message used to get the Light LC Occupancy Mode state of an element. The response to the Light LC OM Get message is a Light LC OM Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_OmSet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_OmSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC OM Set is an acknowledged message used to set the Light LC Occupancy Mode state of an element. The response to the Light LC OM Set message is a Light LC OM Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC OM Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_OmSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LC_CLT_OmSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC OM Set Unacknowledged is an unacknowledged message used to set the Light LC Occupancy Mode state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC OM Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_LightOnOffGet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_LightOnOffGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light LC Light OnOff Get is an acknowledged message used to get the Light LC Light OnOff state of an element. The response to the Light LC Light OnOff Get message is a Light LC Light OnOff Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_LightOnOffSet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_LightOnOffSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Light OnOff Set is an acknowledged message used to set the Light LC Light OnOff state of an element. The response to the Light LC Light OnOff Set message is a Light LC Light OnOff Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Light OnOff Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_LightOnOffSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LC_CLT_LightOnOffSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Light OnOff Set Unacknowledged is an unacknowledged message used to set the Light LC Light OnOff state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Light OnOff Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_PropertyGet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_PropertyGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light LC Property Get is an acknowledged message used to get the Light LC Property state of an element. The response to the Light LC Property Get message is a Light LC Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Property Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_PropertySet()

fsp_err_t RM_MESH_LIGHT_LC_CLT_PropertySet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Property Set is an acknowledged message used to set the Light LC Property state of an element. The response to the Light LC Property Set message is a Light LC Property Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_CLT_PropertySetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_LC_CLT_PropertySetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light LC Property Set Unacknowledged is an unacknowledged message used to set the Light LC Property state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_lc_clt control block. |
| [in] | p_parameter | Pointer to Light LC Property Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Location Server (rm_mesh_light_lc_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_SetTimeProperty (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_device_property_t property, uint32_t time_in_ms) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_SetScenario (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_mesh_light_lc_srv_scenario_t const *const p_scenario) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_GetCurrentScenario (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_mesh_light_lc_srv_scenario_t *const p_scenario) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_ReportOccupancy (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_LC_SRV_ReportLightOnOff (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_mesh_light_lc_srv_light_state_t state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Light Location Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_lc_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_light_lc_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Controller Server (rm_mesh_light_lc_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light Lightness Controller Server (rm_mesh_light_lc_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |

| | | | |
|---|---|---|---|
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness controller server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light lightness controller server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_lc_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_light_lc_srv_scenario_t |
| struct | rm_mesh_light_lc_srv_mode_info_t |
| struct | rm_mesh_light_lc_srv_om_info_t |
| struct | rm_mesh_light_lc_srv_light_onoff_info_t |
| struct | rm_mesh_light_lc_srv_property_id_info_t |
| struct | rm_mesh_light_lc_srv_property_info_t |
| struct | rm_mesh_light_lc_srv_extended_callback_args_t |
| struct | rm_mesh_light_lc_srv_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_light_lc_srv_event_t |
| enum | rm_ble_mesh_light_lc_srv_state_t |
| enum | rm_mesh_light_lc_srv_light_state_t |

## Data Structure Documentation

### ◆ rm_mesh_light_lc_srv_scenario_t

| struct rm_mesh_light_lc_srv_scenario_t | | |
|---|---|---|
| Light LC Server State Info | | |
| Data Fields | | |
| rm_ble_mesh_light_lc_srv_state_t | state | Light LC Server Current Scenario |
| uint32_t | remaining_time_in_ms | Remaining Time in current scenario |

| | | |
|---|---|---|
| uint8_t | occupancy_mode | Light LC Occupancy Mode Value |
| uint8_t | mode | Light LC Mode Value |
| rm_mesh_light_lc_srv_light_state_t | present_light_state | Current Light LC ONOFF State |
| rm_mesh_light_lc_srv_light_state_t | target_light_state | Target Light LC ONOFF State |

### ◆ rm_mesh_light_lc_srv_mode_info_t

| struct rm_mesh_light_lc_srv_mode_info_t | | |
|---|---|---|
| Light LC Mode state | | |
| Data Fields | | |
| uint8_t | present_mode | Light LC Mode state - present |
| uint8_t | target_mode | Light LC Mode state - target |

### ◆ rm_mesh_light_lc_srv_om_info_t

| struct rm_mesh_light_lc_srv_om_info_t | | |
|---|---|---|
| Light LC Occupancy Mode state | | |
| Data Fields | | |
| uint8_t | present_mode | Light LC Occupancy Mode state - present |
| uint8_t | target_mode | Light LC Occupancy Mode state - target |

### ◆ rm_mesh_light_lc_srv_light_onoff_info_t

| struct rm_mesh_light_lc_srv_light_onoff_info_t | | |
|---|---|---|
| Light LC Light OnOff State | | |
| Data Fields | | |
| uint8_t | present_light_onoff | Light LC Light OnOff State |
| uint8_t | target_light_onoff | Light LC Light OnOff State |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_light_lc_srv_property_id_info_t

| struct rm_mesh_light_lc_srv_property_id_info_t |
|---|

| Property ID identifying a Light LC Property | | |
|---|---|---|
| Data Fields | | |
| uint16_t | property_id | Property ID identifying a Light LC Property |

◆ **rm_mesh_light_lc_srv_property_info_t**

| struct rm_mesh_light_lc_srv_property_info_t | | |
|---|---|---|
| Light LC Property state | | |
| Data Fields | | |
| uint16_t | property_id | Property ID identifying a Light LC Property |
| uint8_t * | property_value | Raw value for the Light LC Property |
| uint16_t | property_value_len | |

◆ **rm_mesh_light_lc_srv_extended_callback_args_t**

| struct rm_mesh_light_lc_srv_extended_callback_args_t | | |
|---|---|---|
| Light LC Property state | | |
| Data Fields | | |
| rm_ble_mesh_access_model_handle_t * | p_handle | Access model handle. |
| rm_ble_mesh_light_lc_srv_event_t | event_type | Application events defined for Light LC Server Model. |
| uint8_t * | p_event_data | Event data. |
| uint16_t | event_data_length | Event data length. |

◆ **rm_mesh_light_lc_srv_instance_ctrl_t**

| struct rm_mesh_light_lc_srv_instance_ctrl_t |
|---|
| BLE mesh light lc instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_LC_SRV_Open() is called. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_light_lc_srv_event_t

| enum rm_ble_mesh_light_lc_srv_event_t | |
|---|---|
| Light LC light event | |
| Enumerator | |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_OFF | Light LC Server Event Off. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_STANDBY | Light LC Server Event Standby. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_FADE_ON | Light LC Server Event Fade On. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_RUN | Light LC Server Event Run. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_FADE | Light LC Server Event Fade. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_PROLONG | Light LC Server Event Prolong. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_FADE_STANDBY_AUTO | Light LC Server Event Standby Auto. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_EVENT_FADE_STANDBY_MANUAL | Light LC Server Event Standby Manual. |

### ◆ rm_ble_mesh_light_lc_srv_state_t

| enum rm_ble_mesh_light_lc_srv_state_t | |
|---|---|
| Light LC state | |
| Enumerator | |
| RM_MESH_LIGHT_LC_SRV_STATE_OFF | Light LC Server State Off. |
| RM_MESH_LIGHT_LC_SRV_STATE_STANDBY | Light LC Server State Standby. |
| RM_MESH_LIGHT_LC_SRV_STATE_FADE_ON | Light LC Server State Fade On. |
| RM_MESH_LIGHT_LC_SRV_STATE_RUN | Light LC Server State Run. |
| RM_MESH_LIGHT_LC_SRV_STATE_FADE | Light LC Server State Fade. |
| RM_MESH_LIGHT_LC_SRV_STATE_PROLONG | Light LC Server State Prolong. |
| RM_MESH_LIGHT_LC_SRV_STATE_FADE_STANDBY_AUTO | Light LC Server State Standby Auto. |
| RM_MESH_LIGHT_LC_SRV_STATE_FADE_STANDBY_MANUAL | Light LC Server State Standby Manual. |

### ◆ rm_mesh_light_lc_srv_light_state_t

| enum rm_mesh_light_lc_srv_light_state_t | |
|---|---|
| Light LC light state | |
| Enumerator | |
| RM_MESH_LIGHT_LC_SRV_LIGHT_STATE_OFF | Light state Off. |
| RM_MESH_LIGHT_LC_SRV_LIGHT_STATE_ON | Light state ON. |

## Function Documentation

### ◆ RM_MESH_LIGHT_LC_SRV_Open()

fsp_err_t RM_MESH_LIGHT_LC_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Light_Lc Server model. This is to initialize Light_Lc Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_LC_SRV_Open(&g_mesh_light_lc_srv0_ctrl,
&g_mesh_light_lc_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_LC_SRV_Close()

fsp_err_t RM_MESH_LIGHT_LC_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Light_Lc Server model. This is to terminate Light_Lc Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_LC_SRV_Close(&g_mesh_light_lc_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LC_SRV_StateUpdate()

fsp_err_t RM_MESH_LIGHT_LC_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_LIGHT_LC_SRV_StateUpdate(&g_mesh_light_lc_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_LC_SRV_SetTimeProperty()

fsp_err_t RM_MESH_LIGHT_LC_SRV_SetTimeProperty ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_device_property_t *property*, uint32_t *time_in_ms* )

API to set Light LC Sever Specific Time Properties. This is to update the Light LC Module with any Light Control Time Properties values. Typically these LC Time properties will be needed to be updated or informed to the Light LC Module during Power-Up, Scene Recall/Restore related scenario or when ever the Local application desires to enforce an update in these device properties values. Currently this interface handles only Time related Device Properties of Light LC Module like:
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_OCCUPANCY_DELAY
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_ON
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_RUN_ON
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_PROLONG
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_STANDBY_AUTO
RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_STANDBY_MANUAL

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | property | LC Server Device Property ID |
| [in] | time_in_ms | Related Device Property Value time in milliseconds |

Example:

```
/* Set Light LC Sever Specific Time Properties. */
   err = RM_MESH_LIGHT_LC_SRV_SetTimeProperty(&g_mesh_light_lc_srv0_ctrl, property, time_in_ms);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Set the Light Control Time Properties values successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_MESH_LIGHT_LC_SRV_SetScenario()

fsp_err_t RM_MESH_LIGHT_LC_SRV_SetScenario ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_mesh_light_lc_srv_scenario_t const *const *p_scenario* )

API to locally trigger the Light LC Server state machine from the application. This typically is used in Power-Up or Scene Recall/Restore Scenario where the Application could trigger the Light LC Server from any desired LC State.

Example:

```
/* Trigger to Recall/Restore the scenario. */
   err = RM_MESH_LIGHT_LC_SRV_SetScenario(&g_mesh_light_lc_srv0_ctrl, &scenario);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_scenario | Pointer to rm_mesh_light_lc_srv_scenario_t where Light LC Server State Informations are present |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated scenario successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_scenario is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_MESH_LIGHT_LC_SRV_GetCurrentScenario()

fsp_err_t RM_MESH_LIGHT_LC_SRV_GetCurrentScenario ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_mesh_light_lc_srv_scenario_t *const *p_scenario* )

API to fetch the Light LC Server state machine related information from the Light LC Server Module. This typically is used in Power-Up or Scene Recall/Restore Scenario where the Application uses this to fetch and persistently store the Light LC State related details.

Example:

```
/* Fetch the current scenario. */
   err = RM_MESH_LIGHT_LC_SRV_GetCurrentScenario(&g_mesh_light_lc_srv0_ctrl,
&scenario);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_scenario | Pointer to rm_mesh_light_lc_srv_scenario_t where Light LC Server State Informations are present |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Fetch information successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_scenario is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LC_SRV_ReportOccupancy()

fsp_err_t RM_MESH_LIGHT_LC_SRV_ReportOccupancy ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to report any occupancy to Light LC Server Model.

Example:

```
/* Reports any occupancy. */
   err = RM_MESH_LIGHT_LC_SRV_ReportOccupancy(&g_mesh_light_lc_srv0_ctrl);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_LC_SRV_ReportLightOnOff()

fsp_err_t RM_MESH_LIGHT_LC_SRV_ReportLightOnOff ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_mesh_light_lc_srv_light_state_t *state* )

API to report any Light ON or OFF to Light LC Server Model. There could be many triggering factors to this Light related ON/OFF events from the upper layer. This API could be used for the following scenarios

1. Manual/Physical trigger of Light ON/OFF
2. State Binding with Generic ON/OFF
3. Trigger of Light ON/OFF due to Ambient Light Sensor values

Example:

```
/* Reports any Light ON or OFF. */
   err = RM_MESH_LIGHT_LC_SRV_ReportLightOnOff(&g_mesh_light_lc_srv0_ctrl, state);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | state | Desired state value of Light LC ONOFF. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

## BLE Mesh Network Light Xyl Client (rm_mesh_light_xyl_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_SetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_TargetGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_DefaultGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_DefaultSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_DefaultSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_RangeGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_RangeSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_LIGHT_XYL_CLT_RangeSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Light Xyl Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_xyl_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_light_xyl_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light XYL Client (rm_mesh_light_xyl_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light XYL Client (rm_mesh_light_xyl_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light XYL client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_xyl_clt0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_light_xyl_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_xyl_clt_instance_ctrl_t

| struct rm_mesh_light_xyl_clt_instance_ctrl_t |
|---|
| BLE mesh light xyl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_XYL_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_XYL_CLT_Open()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Light_Xyl Client middleware. This is to initialize Light_Xyl Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_XYL_CLT_Open(&g_mesh_light_xyl_clt0_ctrl,
&g_mesh_light_xyl_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_Close()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Light_Xyl Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_XYL_CLT_Close(&g_mesh_light_xyl_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_XYL_CLT_GetModelHandle()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Light_Xyl client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of light xyl client model. */
   err = RM_MESH_LIGHT_XYL_CLT_GetModelHandle(&g_mesh_light_xyl_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_LIGHT_XYL_CLT_SendReliablePdu()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_LIGHT_XYL_CLT_SendReliablePdu(&g_mesh_light_xyl_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_Get()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light xyL Get is an acknowledged message used to get the Light xyL Lightness, Light xyL x, and Light xyL y states of an element. Upon receiving a Light xyL Get message, the element shall respond with a Light xyL Status message. The response to the Light xyL Get message is a Light xyL Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_Set()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light xyL Set is an acknowledged message used to set the Light xyL Lightness, Light xyL x state, and the Light xyL y states of an element. The response to the Light xyL Set message is a Light xyL Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light xyL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_SetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_SetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

The Light xyL Set Unacknowledged is an unacknowledged message used to set the Light xyL Lightness, Light xyL x, and the Light xyL y states of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light xyL Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_TargetGet()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_TargetGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light xyL Target Get is an acknowledged message used to get the target Light xyL Lightness, Light xyL x, and Light xyL y states of an element. The response to the Light xyL Target Get message is a Light xyL Target Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_DefaultGet()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_DefaultGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Light xyL Default Get is an acknowledged message used to get the Light Lightness Default, the Light xyL x Default, and Light xyL y Default states of an element. The response to the Light xyL Default Get message is a Light xyL Default Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_DefaultSet()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_DefaultSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light xyL Default Set is an acknowledged message used to set the Light Lightness Default, the Light xyL x Default, and Light xyL y Default states of an element. The response to the Light xyL Default Set message is a Light xyL Default Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_DefaultSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_DefaultSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light xyL Default Set Unacknowledged is an unacknowledged message used to set the Light Lightness Default, the Light xyL x Default, and Light xyL y Default states of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light HSL Default Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_RangeGet()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_RangeGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

The Light xyL Range Get is an acknowledged message used to get the Light xyL x Range and Light xyL y Range states of an element. The response to the Light xyL Range Get message is a Light xyL Range Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_RangeSet()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_RangeSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light xyL Range Set is an acknowledged message used to set the Light xyL x Range and Light xyL y Range states of an element. The response to the Light xyL Range Set message is a Light xyL Range Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light xyL Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_LIGHT_XYL_CLT_RangeSetUnacknowledged()

fsp_err_t RM_MESH_LIGHT_XYL_CLT_RangeSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Light xyL Range Set Unacknowledged is an unacknowledged message used to set the Light xyL x Range and Light xyL y Range states of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_light_xyl_clt control block. |
| [in] | p_parameter | Pointer to Light xyL Range Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Light Xyl Server (rm_mesh_light_xyl_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_LIGHT_XYL_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_LIGHT_XYL_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_LIGHT_XYL_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Light Xyl Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_light_xyl_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_light_xyl_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Light XYL Server (rm_mesh_light_xyl_srv)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Light XYL Server (rm_mesh_light_xyl_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light XYL server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh light XYL server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_light_xyl_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_light_xyl_srv_info_t |
| struct | rm_mesh_light_xyl_srv_target_info_t |
| struct | rm_mesh_light_xyl_srv_default_info_t |
| struct | rm_mesh_light_xyl_srv_range_info_t |

| | |
|---|---|
| struct | rm_mesh_light_xyl_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_light_xyl_srv_info_t

struct rm_mesh_light_xyl_srv_info_t

Light xyL state is a composite state that includes the xyL Lightness, the Light xyL x and the Light xyL y states

Data Fields

| | | |
|---|---|---|
| uint16_t | xyl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | target_xyl_lightness | Target perceived lightness - used in response path |
| uint16_t | xyl_x | The 16-bit value representing the x coordinate of a CIE1931 color light |
| uint16_t | target_xyl_x | Target x coordinate - used in response path |
| uint16_t | xyl_y | The 16-bit value representing the y coordinate of a CIE1931 color light |
| uint16_t | target_xyl_y | Target y coordinate - used in response path |
| uint8_t | tid | TID - Used in request path |
| uint8_t | transition_time | Transition Time - Used in request path. Used as remaining time in response path. |
| uint8_t | delay | Delay - Used in request path |
| uint16_t | transition_time_handle | Transition Timer Handle |

### ◆ rm_mesh_light_xyl_srv_target_info_t

struct rm_mesh_light_xyl_srv_target_info_t

Light xyL target state is a composite state that includes the xyL Lightness, the Light xyL x and the Light xyL y states

Data Fields

| | | |
|---|---|---|
| uint16_t | xyl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | xyl_x | The 16-bit value representing the x coordinate of a CIE1931 color light |
| uint16_t | xyl_y | The 16-bit value representing |

| | | |
|---|---|---|
| | | the y coordinate of a CIE1931 color light |

### ◆ rm_mesh_light_xyl_srv_default_info_t

| struct rm_mesh_light_xyl_srv_default_info_t | | |
|---|---|---|
| Light xyL default state is a composite state that includes the xyL Lightness, the Light xyL x and the Light xyL y states | | |
| Data Fields | | |
| uint16_t | xyl_lightness | The perceived lightness of a light emitted by the element |
| uint16_t | xyl_x | The 16-bit value representing the x coordinate of a CIE1931 color light |
| uint16_t | xyl_y | The 16-bit value representing the y coordinate of a CIE1931 color light |

### ◆ rm_mesh_light_xyl_srv_range_info_t

| struct rm_mesh_light_xyl_srv_range_info_t | | |
|---|---|---|
| Light xyL Range state determines the minimum and maximum values of the Light xyL x and syL y state of an element | | |
| Data Fields | | |
| uint16_t | xyl_x_range_min | The minimum value of a Light xyL x state of an element |
| uint16_t | xyl_x_range_max | The maximum value of a Light xyL x state of an element |
| uint16_t | xyl_y_range_min | The minimum value of a Light xyL y state of an element |
| uint16_t | xyl_y_range_max | The maximum value of a Light xyL y state of an element |
| uint8_t | status | Status - Used in the response path |

### ◆ rm_mesh_light_xyl_srv_instance_ctrl_t

| struct rm_mesh_light_xyl_srv_instance_ctrl_t |
|---|
| BLE mesh light xyl instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_LIGHT_XYL_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_LIGHT_XYL_SRV_Open()

fsp_err_t RM_MESH_LIGHT_XYL_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Light_Xyl Server model. This is to initialize Light_Xyl Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_LIGHT_XYL_SRV_Open(&g_mesh_light_xyl_srv0_ctrl,
&g_mesh_light_xyl_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_LIGHT_XYL_SRV_Close()

fsp_err_t RM_MESH_LIGHT_XYL_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Light_Xyl Server model. This is to terminate Light_Xyl Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_LIGHT_XYL_SRV_Close(&g_mesh_light_xyl_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_LIGHT_XYL_SRV_StateUpdate()

fsp_err_t RM_MESH_LIGHT_XYL_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_LIGHT_XYL_SRV_StateUpdate(&g_mesh_light_xyl_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Lower Trans (rm_ble_mesh_lower_trans)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_Open (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_Close (rm_ble_mesh_provision_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_SendPdu |

| | |
|---|---|
| | (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_transmit_setting_t const *const p_transmit_setting, rm_ble_mesh_buffer_t const *const p_buffer, rm_ble_mesh_lower_trans_reliable_t reliable) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_ClearSarContexts (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_ClearSubnetSarContexts (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_ReinitReplayCache (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_LOWER_TRANS_TriggerPendingTransmits (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Lower Trans module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_lower_trans

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_lower_trans_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Lower Trans (rm_ble_mesh_lower_trans)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Lower Trans (rm_ble_mesh_lower_trans).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_lower_trans0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the |

| | | | |
|---|---|---|---|
| | | | hardware. |
| Notification Enable | • Enabled<br>• Disabled | Disabled | Whether to enable the notification or not. |
| Callback | Name Must Be a Valid C Symbol | NULL | Callback function name. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_lower_trans_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_lower_trans_instance_ctrl_t

| struct rm_ble_mesh_lower_trans_instance_ctrl_t |
|---|
| RM_BLE_MESH_LOWER_TRANS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_LOWER_TRANS_Open() is called. |

## Function Documentation

### ◆ RM_BLE_MESH_LOWER_TRANS_Open()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_Open ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_lower_trans_cfg_t const *const *p_cfg* )

Register Inerface with Lower Transport Layer. This routine registers interface with the Lower Transport Layer. Transport Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_lower_trans_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_LOWER_TRANS_Open(&g_ble_mesh_lower_trans0_ctrl,
&g_ble_mesh_lower_trans0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_LOWER_TRANS_Close()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_Close ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*)

Unregister Inerface with Lower Transport Layer. Implements rm_ble_mesh_lower_trans_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_LOWER_TRANS_Close(&g_ble_mesh_lower_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_LOWER_TRANS_SendPdu()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_SendPdu ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_lower_trans_transmit_setting_t const *const *p_transmit_setting*, rm_ble_mesh_buffer_t const *const *p_buffer*, rm_ble_mesh_lower_trans_reliable_t *reliable* )

API to send transport PDUs. This routine sends transport PDUs to peer device.

Implements rm_ble_mesh_lower_trans_api_t::sendPdu.

Example:

```
/* Send transport PDUs. */
   err = RM_BLE_MESH_LOWER_TRANS_SendPdu(&g_ble_mesh_lower_trans0_ctrl,
                                         &transmit_setting,
                                         &buffer,
RM_BLE_MESH_LOWER_TRANS_RELIABLE_ENABLE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_transmit_setting and p_buffer are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_LOWER_TRANS_ClearSarContexts()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_ClearSarContexts ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*)

To clear all segmentation and reassembly contexts. This routine clears all segmentation and reassembly contexts.

Implements rm_ble_mesh_lower_trans_api_t::clearSarContexts.

Example:

```
/* Clear all segmentation and reassembly contexts. */
   err = RM_BLE_MESH_LOWER_TRANS_ClearSarContexts(&g_ble_mesh_lower_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_LOWER_TRANS_ClearSubnetSarContexts()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_ClearSubnetSarContexts ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle* )

To clear all segmentation and reassembly contexts for a given subnet. This routine clears all segmentation and reassembly contexts.

Implements rm_ble_mesh_lower_trans_api_t::clearSubnetSarContexts.

Example:

```
/* Clear all segmentation and reassembly contexts for a given subnet. */
   err =
RM_BLE_MESH_LOWER_TRANS_ClearSubnetSarContexts(&g_ble_mesh_lower_trans0_ctrl,
subnet_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_LOWER_TRANS_ReinitReplayCache()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_ReinitReplayCache ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*)

To reinitialize all Lower Transport replay cache entries. This routine clears and reinitializes all Transport Replay Cache Entries. This needs to be invoked by the upper layer when the Network moves to a newer IV Index (Normal State) and the Sequence numbers are reset.

Implements rm_ble_mesh_lower_trans_api_t::reinitReplayCache.

Example:

```
/* Reinitialize all Lower Transport replay cache Entries. */
   err = RM_BLE_MESH_LOWER_TRANS_ReinitReplayCache(&g_ble_mesh_lower_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_LOWER_TRANS_TriggerPendingTransmits()

fsp_err_t RM_BLE_MESH_LOWER_TRANS_TriggerPendingTransmits ( rm_ble_mesh_lower_trans_ctrl_t *const *p_ctrl*)

To trigger any LTRN pending transmissions. Trigger pending transmits is an interface to check for pending segments in the tx queue and schedule for transmission, which is mainly used by the LPN operation.

Implements rm_ble_mesh_lower_trans_api_t::triggerPendingTransmits.

Example:

```
/* Trigger any Lower Transport pending transmissions. */
   err =
RM_BLE_MESH_LOWER_TRANS_TriggerPendingTransmits(&g_ble_mesh_lower_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Network (rm_ble_mesh_network)

Modules » Networking » BLE Mesh Network Modules

## Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_NETWORK_Open (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_NETWORK_Close (rm_ble_mesh_network_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_NETWORK_BroadcastSecureBeacon (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| fsp_err_t | RM_BLE_MESH_NETWORK_SendPduOnInterface (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_network_header_t const *const p_header, rm_ble_mesh_buffer_t const *const p_buffer) |
| fsp_err_t | RM_BLE_MESH_NETWORK_GetAddressType (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_network_address_type_t *const p_type) |
| fsp_err_t | RM_BLE_MESH_NETWORK_FetchProxyState (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_gatt_proxy_state_t *const p_proxy_state) |
| fsp_err_t | RM_BLE_MESH_NETWORK_SetProxyFilter (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_filter_type_t type) |
| fsp_err_t | RM_BLE_MESH_NETWORK_ConfigProxyFilter (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_config_opcode_t opcode, rm_ble_mesh_network_proxy_address_list_t *const p_addr_list) |
| fsp_err_t | RM_BLE_MESH_NETWORK_StartProxyServerAdv (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, rm_ble_mesh_network_gatt_proxy_adv_mode_t proxy_adv_mode) |
| fsp_err_t | RM_BLE_MESH_NETWORK_StopProxyServerAdv (rm_ble_mesh_network_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_NETWORK_AllocateSeqNumber (rm_ble_mesh_network_ctrl_t *const p_ctrl, uint32_t *const p_seq_num) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_NETWORK_GetSeqNumberState (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t *const p_seq_num_state) |
| fsp_err_t | RM_BLE_MESH_NETWORK_SetSeqNumberState (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t const *const p_seq_num_state) |
| fsp_err_t | RM_BLE_MESH_NETWORK_ResetNetCache (rm_ble_mesh_network_ctrl_t *const p_ctrl) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Network module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_network

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_network_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Network (rm_ble_mesh_network)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Network (rm_ble_mesh_network).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_network0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Ignore Netcache Wrapping | • Processed<br>• Dropped | Dropped | Ignore netcache wrapping. |
| RX Callback Event Enable | • Enabled<br>• Disabled | Disabled | Whether to enable the RX callback event or |

| | | | |
|---|---|---|---|
| | | | not. |
| TX Callback Event Enable | • Enabled<br>• Disabled | Disabled | Whether to enable the TX callback event or not. |
| Callback | Name Must Be a Valid C Symbol | NULL | Callback function name. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_network_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_network_instance_ctrl_t

struct rm_ble_mesh_network_instance_ctrl_t

RM_BLE_MESH_NETWORK private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_NETWORK_Open() is called.

## Function Documentation

### ◆ RM_BLE_MESH_NETWORK_Open()

fsp_err_t RM_BLE_MESH_NETWORK_Open ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_cfg_t const *const *p_cfg* )

Register Inerface with Network Layer. This routine registers interface with the Network Layer. Network Layer supports only one upper layer, hence this routine shall be called once.

Implements rm_ble_mesh_network_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_NETWORK_Open(&g_ble_mesh_network0_ctrl,
&g_ble_mesh_network0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ RM_BLE_MESH_NETWORK_Close()

fsp_err_t RM_BLE_MESH_NETWORK_Close ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*)

Unregister Inerface with Network Layer. Implements rm_ble_mesh_network_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_NETWORK_Close(&g_ble_mesh_network0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_BroadcastSecureBeacon()

fsp_err_t RM_BLE_MESH_NETWORK_BroadcastSecureBeacon ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle* )

API to send Secure Network Beacon. This routine sends Secure Network Beacon for the given subnet handle.

Implements rm_ble_mesh_network_api_t::broadcastSecureBeacon.

Example:

```
/* Send secure network beacon. */
   err = RM_BLE_MESH_NETWORK_BroadcastSecureBeacon(&g_ble_mesh_network0_ctrl,
subnet_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_NETWORK_SendPduOnInterface()

fsp_err_t RM_BLE_MESH_NETWORK_SendPduOnInterface ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_route_info_t const *const *p_route_info*, rm_ble_mesh_network_header_t const *const *p_header*, rm_ble_mesh_buffer_t const *const *p_buffer* )

Extension API to send Network PDUs on selected network interfaces. This routine sends NETWORK PDUs on all or selected Network Interfaces.

Implements rm_ble_mesh_network_api_t::sendPduOnInterface.

Example:

```
/* Send network PDUs on selected network interfaces. */
   err = RM_BLE_MESH_NETWORK_SendPduOnInterface(&g_ble_mesh_network0_ctrl,
&route_info, &header, &buffer);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_route_info, p_header and p_buffer are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_NETWORK_GetAddressType()

fsp_err_t RM_BLE_MESH_NETWORK_GetAddressType ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr*, rm_ble_mesh_network_address_type_t *const *p_type* )

To get address type. This routine is to get address type for a given address.

Implements rm_ble_mesh_network_api_t::getAddressType.

Example:

```
/* Get address type. */
   err = RM_BLE_MESH_NETWORK_GetAddressType(&g_ble_mesh_network0_ctrl, addr, &type);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_type is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_FetchProxyState()

fsp_err_t RM_BLE_MESH_NETWORK_FetchProxyState ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_gatt_proxy_state_t *const *p_proxy_state* )

Check if the proxy module is ready to handle proxy messages/events. This routine returns the current state of the Proxy. The valid states of proxy are:

1. RM_BLE_MESH_NETWORK_GATT_PROXY_STATE_NULL - If no callback registered by Upper Layers
2. RM_BLE_MESH_NEgTWORK_GATT_PROXY_STATE_READY - If callback registered and Proxy not connected
3. RM_BLE_MESH_NETWORK_GATT_PROXY_STATE_CONNECTED - if callback registered and Proxy connected

Implements rm_ble_mesh_network_api_t::fetchProxyState.

Example:

```
/* Check if the proxy module is ready to handle proxy messages/events. */
   err = RM_BLE_MESH_NETWORK_FetchProxyState(&g_ble_mesh_network0_ctrl,
&proxy_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_proxy_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_SetProxyFilter()

fsp_err_t RM_BLE_MESH_NETWORK_SetProxyFilter ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_route_info_t const *const *p_route_info*, rm_ble_mesh_proxy_filter_type_t *type* )

Set proxy server's filter type. This function is used by the Proxy Client to set the filter type on the Proxy Server.

Implements rm_ble_mesh_network_api_t::setProxyFilter.

Example:

```
/* Set proxy server's filter type. */
   err = RM_BLE_MESH_NETWORK_SetProxyFilter(&g_ble_mesh_network0_ctrl, &route_info,
filter_type);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_route_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_ConfigProxyFilter()

fsp_err_t RM_BLE_MESH_NETWORK_ConfigProxyFilter ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_route_info_t const *const *p_route_info*, rm_ble_mesh_proxy_config_opcode_t *opcode*, rm_ble_mesh_network_proxy_address_list_t *const *p_addr_list* )

Add or Delete/Remove addresses to/from proxy filter list. This function is used by the Proxy Client to add/delete Addresses to/from the Proxy Server's filter List.

Implements rm_ble_mesh_network_api_t::configProxyFilter.

Example:

```
/* Add or delete/remove addresses to/from proxy filter list. */
   err = RM_BLE_MESH_NETWORK_ConfigProxyFilter(&g_ble_mesh_network0_ctrl,
&route_info, opcode, &addr_list);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_route_info and p_addr_list are NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_StartProxyServerAdv()

fsp_err_t RM_BLE_MESH_NETWORK_StartProxyServerAdv ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, rm_ble_mesh_network_gatt_proxy_adv_mode_t *proxy_adv_mode* )

Start connectable advertisements for a proxy server. This function is used by the Proxy Server to start Connectable Undirected Advertisements.

Implements rm_ble_mesh_network_api_t::startProxyServerAdv.

Example:

```
/* Start connectable advertisements for a proxy server. */
   err = RM_BLE_MESH_NETWORK_StartProxyServerAdv(&g_ble_mesh_network0_ctrl,
subnet_handle, proxy_adv_mode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The internal parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_NETWORK_StopProxyServerAdv()

fsp_err_t RM_BLE_MESH_NETWORK_StopProxyServerAdv ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*)

Stop connectable advertisements for a proxy server. This function is used by the Proxy Server to stop Connectable Undirected Advertisements.

Implements rm_ble_mesh_network_api_t::stopProxyServerAdv.

Example:

```
/* Stop connectable advertisements for a proxy server. */
   err = RM_BLE_MESH_NETWORK_StopProxyServerAdv(&g_ble_mesh_network0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_AllocateSeqNumber()

fsp_err_t RM_BLE_MESH_NETWORK_AllocateSeqNumber ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, uint32_t *const *p_seq_num* )

To allocate sequence number. This function is used to allocate Sequence Number.

Implements rm_ble_mesh_network_api_t::allocateSeqNumber.

Example:

```
/* Allocate sequence number. */
   err = RM_BLE_MESH_NETWORK_AllocateSeqNumber(&g_ble_mesh_network0_ctrl, &seq_num);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_seq_num is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_GetSeqNumberState()

fsp_err_t RM_BLE_MESH_NETWORK_GetSeqNumberState ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_seq_number_state_t *const *p_seq_num_state* )

To get current sequence number state. This function is used to get current Sequence Number state.

Implements rm_ble_mesh_network_api_t::getSeqNumberState.

Example:

```
/* Get current sequence number state. */
   err = RM_BLE_MESH_NETWORK_GetSeqNumberState(&g_ble_mesh_network0_ctrl,
&seq_num_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_seq_num_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_SetSeqNumberState()

fsp_err_t RM_BLE_MESH_NETWORK_SetSeqNumberState ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_seq_number_state_t const *const *p_seq_num_state* )

To set current sequence number state. This function is used to get current Sequence Number state, which acquiring lock. Used from persistent storage.

Implements rm_ble_mesh_network_api_t::setSeqNumberState.

Example:

```
/* Set current sequence number state. */
   err = RM_BLE_MESH_NETWORK_SetSeqNumberState(&g_ble_mesh_network0_ctrl,
&seq_num_state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_seq_num_state is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_NETWORK_ResetNetCache()

fsp_err_t RM_BLE_MESH_NETWORK_ResetNetCache ( rm_ble_mesh_network_ctrl_t *const *p_ctrl*)

To reinitialize all Network Layer cache entries. This routine clears and reinitializes all Network Cache Entries. This needs to be invoked by the upper layer when the Network moves to a newer IV Index (Normal State) and the Sequence numbers are reset.

Implements rm_ble_mesh_network_api_t::resetNetCache.

Example:

```
/* Reinitialize all Network Layer cache entries. */
   err = RM_BLE_MESH_NETWORK_ResetNetCache(&g_ble_mesh_network0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## BLE Mesh Network Provision (rm_ble_mesh_provision)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_PROVISION_Open (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_PROVISION_Close (rm_ble_mesh_provision_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_PROVISION_Setup (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_role_t role, rm_ble_mesh_provision_device_info_t info, uint16_t timeout) |
| fsp_err_t | RM_BLE_MESH_PROVISION_Bind (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_device_info_t info, uint8_t attention, rm_ble_mesh_provision_handle_t *const p_handle) |
| fsp_err_t | RM_BLE_MESH_PROVISION_SendPdu (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_pdu_type_t type, rm_ble_mesh_buffer_t pdu_data) |
| fsp_err_t | RM_BLE_MESH_PROVISION_SetAuthVal (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_buffer_t auth_value) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_PROVISION_Abort (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_link_close_reason_t reason) |
| fsp_err_t | RM_BLE_MESH_PROVISION_GetLocalPublicKey (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_public_key) |
| fsp_err_t | RM_BLE_MESH_PROVISION_SetLocalPublicKey (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_public_key) |
| fsp_err_t | RM_BLE_MESH_PROVISION_GenerateRandomizedNumber (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_key) |
| fsp_err_t | RM_BLE_MESH_PROVISION_SetOobPublicKey (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_key, uint8_t size) |
| fsp_err_t | RM_BLE_MESH_PROVISION_SetOobAuthInfo (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_auth_info, uint8_t size) |
| fsp_err_t | RM_BLE_MESH_PROVISION_GenerateEcdhKey (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_public_key) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Provision module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_provision

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_provision_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Provision (rm_ble_mesh_provision)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Provision (rm_ble_mesh_provision).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_provision0 | Module name. |
| Provision Capabilities | | | |
| Number of Elements | Invalid Number of Elements | 1 | Provision capabilities number of elements. |
| Supported Algorithms | Invalid Supported Algorithms | 0 | Provision capabilities supported algorithms. |
| Public Key Type | Invalid Public Key Type | 0 | Provision capabilities public key type. |
| Static OOB Type | Invalid Static OOB Type | 0 | Provision capabilities static OOB type. |
| Output OOB Action | Invalid Output OOB Action | 0 | Provision capabilities output OOB action. |
| Output OOB Size | Invalid Output OOB Size | 0 | Provision capabilities output OOB size. |
| Input OOB Action | Invalid Input OOB Action | 0 | Provision capabilities input OOB action. |
| Input OOB Size | Invalid Input OOB Size | 0 | Provision capabilities iutput OOB size. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Provision Callback | Name Must Be a Valid C Symbol. | NULL | Provision callback function name. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_provision_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_provision_instance_ctrl_t

| struct rm_ble_mesh_provision_instance_ctrl_t |
|---|
| RM_BLE_MESH_PROVISION private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_PROVISION_Open() is called. |

## Function Documentation

### ◆ RM_BLE_MESH_PROVISION_Open()

fsp_err_t RM_BLE_MESH_PROVISION_Open ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_cfg_t const *const *p_cfg* )

Open access middleware. This function registers the provisioning capabilities of the application along with the application callback to notify events during the provisioning procedure.

Implements rm_ble_mesh_provision_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_PROVISION_Open(&g_ble_mesh_provision0_ctrl,
&g_ble_mesh_provision0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_PROVISION_Close()

fsp_err_t RM_BLE_MESH_PROVISION_Close ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*)

Close access middleware. Implements rm_ble_mesh_provision_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_PROVISION_Close(&g_ble_mesh_provision0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_PROVISION_Setup()

fsp_err_t RM_BLE_MESH_PROVISION_Setup ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_role_t *role*, rm_ble_mesh_provision_device_info_t *info*, uint16_t *timeout* )

Setup the device for provisioning. This function configures the device to get in a provisionable state by specifying the role, bearer and creating a context.

Implements rm_ble_mesh_provision_api_t::setup.

Example:

```
/* Setup the device for provisioning. */
   err = RM_BLE_MESH_PROVISION_Setup(&g_ble_mesh_provision0_ctrl, role, info,
timeout);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | The parameter info->p_device is NULL. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_PROVISION_Bind()

fsp_err_t RM_BLE_MESH_PROVISION_Bind ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_device_info_t *info*, uint8_t *attention*, rm_ble_mesh_provision_handle_t *const *p_handle* )

Bind to the peer device for provisioning. This function establishes a provisioning link with the peer device and exchanges the capabilities for provisioning.

Implements rm_ble_mesh_provision_api_t::bind.

Example:

```
/* Bind to the peer device for provisioning. */
   err = RM_BLE_MESH_PROVISION_Bind(&g_ble_mesh_provision0_ctrl, info, attention,
&provision_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_PROVISION_SendPdu()

fsp_err_t RM_BLE_MESH_PROVISION_SendPdu ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_handle_t const *const *p_handle*, rm_ble_mesh_provision_pdu_type_t *type*, rm_ble_mesh_buffer_t *pdu_data* )

Send provisioning PDUs to the peer. This function is used by the provisioning application to send the provisioning PDUs to the peer device during the procedure.

Implements rm_ble_mesh_provision_api_t::sendPdu.

Example:

```
/* Send provisioning PDUs to the peer. */
   err = RM_BLE_MESH_PROVISION_SendPdu(&g_ble_mesh_provision0_ctrl,
&provision_handle, type, pdu_data);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_UNSUPPORTED | The pdu type is not supported. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_PROVISION_SetAuthVal()

fsp_err_t RM_BLE_MESH_PROVISION_SetAuthVal ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, rm_ble_mesh_provision_handle_t const *const *p_handle*, rm_ble_mesh_buffer_t *auth_value* )

Set the display authval. This function shall be used by the provisioning application to set the authval being displayed to the user on receiving RM_BLE_MESH_PROVISION_EVENT_TYPE_OOB_DISPLAY event with the respective OOB Action and Size.

Implements rm_ble_mesh_provision_api_t::setAuthVal.

Example:

```
rm_ble_mesh_buffer_t auth_value = {.payload = NULL, .length = 0};;

/* Set the display authval. */

   err = RM_BLE_MESH_PROVISION_SetAuthVal(&g_ble_mesh_provision0_ctrl,
&provision_handle, auth_value);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_PROVISION_Abort()

fsp_err_t RM_BLE_MESH_PROVISION_Abort ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*,
rm_ble_mesh_provision_handle_t const *const *p_handle*,
rm_ble_mesh_provision_link_close_reason_t *reason* )

Abort the provisioning procedure. This API can be used by the application to abort the ongoing provisioning procedure. This routine closes the provisioning link with the reason as specified.

Implements rm_ble_mesh_provision_api_t::abort.

Example:

```
/* Abort the provisioning procedure. */
   err = RM_BLE_MESH_PROVISION_Abort(&g_ble_mesh_provision0_ctrl, &provision_handle,
reason);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_PROVISION_GetLocalPublicKey()

fsp_err_t RM_BLE_MESH_PROVISION_GetLocalPublicKey ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t *const *p_public_key* )

Utility API to get current ECDH public key to be used for provisioning. This API can be used by the application to fetch the current ECDH P256 Public Key which is to be used for the Provisioning Procedure.

Implements rm_ble_mesh_provision_api_t::getLocalPublicKey.

Example:

```
/* Get current ECDH Public Key to be used for Provisioning. */
   err = RM_BLE_MESH_PROVISION_GetLocalPublicKey(&g_ble_mesh_provision0_ctrl,
&public_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_public_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_PROVISION_SetLocalPublicKey()

fsp_err_t RM_BLE_MESH_PROVISION_SetLocalPublicKey ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t const *const *p_public_key* )

Utility API to set current ECDH public key to be used for provisioning. This API can be used by the application to fetch the current ECDH P256 Public Key which is to be used for the Provisioning Procedure.

Implements rm_ble_mesh_provision_api_t::setLocalPublicKey.

Example:

```
/* Set current ECDH Public Key to be used for Provisioning. */
   err = RM_BLE_MESH_PROVISION_SetLocalPublicKey(&g_ble_mesh_provision0_ctrl,
&public_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_public_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_PROVISION_GenerateRandomizedNumber()

fsp_err_t RM_BLE_MESH_PROVISION_GenerateRandomizedNumber ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t *const *p_key* )

Utility API to generate 128bits (16 bytes) randomized number to be used for provisioning. The randomized number can be used for UUID, Net Key and Application Key.

Implements rm_ble_mesh_provision_api_t::generateRandomizedNumber.

Example:

```
/* Generate 128bits (16 bytes) randomized number. */
   err = RM_BLE_MESH_PROVISION_GenerateRandomizedNumber(&g_ble_mesh_provision0_ctrl,
p_net_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_PROVISION_SetOobPublicKey()

fsp_err_t RM_BLE_MESH_PROVISION_SetOobPublicKey ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t const *const *p_key*, uint8_t *size* )

Utility API to set device out of band public key for provisioning.

Implements rm_ble_mesh_provision_api_t::setOobPublicKey.

Example:

```
/* Set device out of band public key. */
   err = RM_BLE_MESH_PROVISION_SetOobPublicKey(&g_ble_mesh_provision0_ctrl,
p_public_key, size);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_key is NULL. |
| FSP_ERR_INVALID_ARGUMENT | The parameter size is more than 64. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_PROVISION_SetOobAuthInfo()

fsp_err_t RM_BLE_MESH_PROVISION_SetOobAuthInfo ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t const *const *p_auth_info*, uint8_t *size* )

Utility API to set device out of band authentication information for provisioning.

Implements rm_ble_mesh_provision_api_t::setOobAuthInfo.

Example:

```
/* Set device out of band authentication information. */
   err = RM_BLE_MESH_PROVISION_SetOobAuthInfo(&g_ble_mesh_provision0_ctrl,
p_auth_info, size);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_auth_info is NULL. |
| FSP_ERR_INVALID_ARGUMENT | The parameter size is more than 16. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_PROVISION_GenerateEcdhKey()

fsp_err_t RM_BLE_MESH_PROVISION_GenerateEcdhKey ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*, uint8_t *const *p_public_key* )

Utility API to generate ECDH public key to be used for provisioning.

Implements rm_ble_mesh_provision_api_t::generateEcdhKey.

Example:

```
/* Generate ECDH Public Key for Provisioning. */
   err = RM_BLE_MESH_PROVISION_GenerateEcdhKey(&g_ble_mesh_provision0_ctrl,
&public_key);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_public_key is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

# BLE Mesh Network Scene Client (rm_mesh_scene_clt)

Modules » Networking » BLE Mesh Network Modules

## Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SCENE_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SCENE_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCENE_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_SCENE_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_SCENE_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCENE_CLT_Recall (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCENE_CLT_RecallUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCENE_CLT_RegisterGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCENE_CLT_Store (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCENE_CLT_StoreUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCENE_CLT_Delete (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCENE_CLT_DeleteUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

## Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Scene Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_scene_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_scene_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Scene Client (rm_mesh_scene_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Scene Client (rm_mesh_scene_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scene client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_scene_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_scene_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_scene_clt_instance_ctrl_t

| struct rm_mesh_scene_clt_instance_ctrl_t |
|---|
| BLE mesh scene instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SCENE_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_SCENE_CLT_Open()

fsp_err_t RM_MESH_SCENE_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

open Scene Client middleware. This is to initialize Scene Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_SCENE_CLT_Open(&g_mesh_scene_clt0_ctrl, &g_mesh_scene_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_SCENE_CLT_Close()

fsp_err_t RM_MESH_SCENE_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Scene Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SCENE_CLT_Close(&g_mesh_scene_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SCENE_CLT_GetModelHandle()

fsp_err_t RM_MESH_SCENE_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Scene client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of scene client model. */
   err = RM_MESH_SCENE_CLT_GetModelHandle(&g_mesh_scene_clt0_ctrl, &model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_SCENE_CLT_SendReliablePdu()

fsp_err_t RM_MESH_SCENE_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_SCENE_CLT_SendReliablePdu(&g_mesh_scene_clt0_ctrl, req_opcode,
p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_Get()

fsp_err_t RM_MESH_SCENE_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Scene Get is an acknowledged message used to get the current status of a currently active scene of an element. The response to the Scene Get message is a Scene Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_Recall()

fsp_err_t RM_MESH_SCENE_CLT_Recall ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Recall is an acknowledged message that is used to recall the current state of an element from a previously stored scene. The response to the Scene Recall message is a Scene Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Recall message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_RecallUnacknowledged()

fsp_err_t RM_MESH_SCENE_CLT_RecallUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Recall Unacknowledged is an unacknowledged message used to recall the current state of an element from a previously stored Scene.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Recall message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_RegisterGet()

fsp_err_t RM_MESH_SCENE_CLT_RegisterGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Scene Register Get is an acknowledged message used to get the current status of the Scene Register of an element. The response to the Scene Register Get message is a Scene Register Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_Store()

fsp_err_t RM_MESH_SCENE_CLT_Store ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Store is an acknowledged message used to store the current state of an element as a Scene, which can be recalled later. The response to the Scene Store message is a Scene Register Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Store message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_StoreUnacknowledged()

fsp_err_t RM_MESH_SCENE_CLT_StoreUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Store Unacknowledged is an unacknowledged message used to store the current state of an element as a Scene, which can be recalled later.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Store message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_Delete()

fsp_err_t RM_MESH_SCENE_CLT_Delete ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Delete is an acknowledged message used to delete a Scene from the Scene Register state of an element. The response to the Scene Delete message is a Scene Register Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Delete parameter. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCENE_CLT_DeleteUnacknowledged()

fsp_err_t RM_MESH_SCENE_CLT_DeleteUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scene Delete Unacknowledged is an unacknowledged message used to delete a scene from the Scene Register state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scene_clt control block. |
| [in] | p_parameter | Pointer to Scene Delete parameter. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Scene Server (rm_mesh_scene_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SCENE_SRV_Open (rm_ble_mesh_scene_server_ctrl_t *const p_ctrl, rm_ble_mesh_scene_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SCENE_SRV_Close (rm_ble_mesh_scene_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCENE_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_scene_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Scene Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_scene_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_scene_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Scene Server (rm_mesh_scene_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Scene Server (rm_mesh_scene_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scene server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scene server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_scene_srv0 | Module name. |
| Model Handle | Invalid Model Handle | 0 | Select model handle. |
| Setup Server Handle | Invalid Setup Server Handle | 0 | Select setup server handle. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_scene_srv_number_info_t |
| struct | rm_mesh_scene_srv_status_info_t |

| | |
|---|---|
| struct | rm_mesh_scene_srv_register_status_info_t |
| struct | rm_mesh_scene_srv_ext_tid_and_transition_info_t |
| struct | rm_mesh_scene_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_scene_srv_number_info_t

| struct rm_mesh_scene_srv_number_info_t | | |
|---|---|---|
| The state to identify a scene | | |
| Data Fields | | |
| uint16_t | number | The number to identify a scene |

### ◆ rm_mesh_scene_srv_status_info_t

| struct rm_mesh_scene_srv_status_info_t | | |
|---|---|---|
| The current status of a currently active scene | | |
| Data Fields | | |
| uint8_t | status_code | Status Code |
| uint16_t | current_scene | Scene Number of a current scene. |
| uint16_t | target_scene | Scene Number of a target scene. |
| uint8_t | remaining_time | |

### ◆ rm_mesh_scene_srv_register_status_info_t

| struct rm_mesh_scene_srv_register_status_info_t | | |
|---|---|---|
| The current status of scene register | | |
| Data Fields | | |
| uint8_t | status_code | Status Code |
| uint16_t | current_scene | Scene Number of a current scene. |
| uint16_t * | scenes | A list of scenes stored within an element |
| uint16_t | scenes_count | |

### ◆ rm_mesh_scene_srv_ext_tid_and_transition_info_t

| struct rm_mesh_scene_srv_ext_tid_and_transition_info_t |
|---|
| TID and Transition is a structure which contains Transaction ID (TID) as mandatory field. Other two fields, Transition Time and Delay are optional. |

TID field is a transaction identifier indicating whether the message is a new message or a retransmission of a previously sent message.

If present, the Transition Time field identifies the time that an element will take to transition to the target state from the present state.

The Delay field shall be present when the Transition Time field is present. It identifies the message execution delay, representing a time interval between receiving the message by a model and executing the associated model behaviors.

### ◆ rm_mesh_scene_srv_instance_ctrl_t

| struct rm_mesh_scene_srv_instance_ctrl_t |
|---|
| BLE mesh scene instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SCENE_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_SCENE_SRV_Open()

| fsp_err_t RM_MESH_SCENE_SRV_Open ( rm_ble_mesh_scene_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_scene_server_cfg_t const *const *p_cfg* ) |
|---|
| API to initialize Scene Server model. This is to initialize Scene Server model and to register with Access layer.<br>Implements rm_ble_mesh_model_server_api_t::open.<br>Example:<br>`/* Open the module. */`<br>`err = RM_MESH_SCENE_SRV_Open(&g_mesh_scene_srv0_ctrl, &g_mesh_scene_srv0_cfg);` |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_SCENE_SRV_Close()

fsp_err_t RM_MESH_SCENE_SRV_Close ( rm_ble_mesh_scene_server_ctrl_t *const *p_ctrl*)

API to terminate Scene Server model. This is to terminate Scene Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SCENE_SRV_Close(&g_mesh_scene_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SCENE_SRV_StateUpdate()

fsp_err_t RM_MESH_SCENE_SRV_StateUpdate ( rm_ble_mesh_scene_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_SCENE_SRV_StateUpdate(&g_mesh_scene_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Scheduler Client (rm_mesh_scheduler_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SCHEDULER_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_ActionGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_ActionSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SCHEDULER_CLT_ActionSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Scheduler Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_scheduler_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_scheduler_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

**Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Scheduler Client (rm_mesh_scheduler_clt)**

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Scheduler Client (rm_mesh_scheduler_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scheduler |

| | | | |
|---|---|---|---|
| | | | client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_scheduler_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_scheduler_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_scheduler_clt_instance_ctrl_t

| |
|---|
| struct rm_mesh_scheduler_clt_instance_ctrl_t |
| BLE mesh scheduler instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SCHEDULER_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_SCHEDULER_CLT_Open()

| |
|---|
| fsp_err_t RM_MESH_SCHEDULER_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* ) |
| Open Scheduler Client middleware. This is to initialize Scheduler Client model and to register with Access layer.<br>Implements rm_ble_mesh_model_client_api_t::open.<br>Example:<br>`/* Open the module. */`<br>`err = RM_MESH_SCHEDULER_CLT_Open(&g_mesh_scheduler_clt0_ctrl, &g_mesh_scheduler_clt0_cfg);`<br>**Return values**<br>FSP_SUCCESS — Model opened successfully.<br>FSP_ERR_ASSERTION — Pointer to control block or configuration structure is NULL.<br>FSP_ERR_ALREADY_OPEN — Model is already open.<br>FSP_ERR_NOT_FOUND — The number of models has exceeded the limit.<br>FSP_ERR_ABORTED — Model initialization is failed. |

### ◆ RM_MESH_SCHEDULER_CLT_Close()

fsp_err_t RM_MESH_SCHEDULER_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Scheduler Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SCHEDULER_CLT_Close(&g_mesh_scheduler_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SCHEDULER_CLT_GetModelHandle()

fsp_err_t RM_MESH_SCHEDULER_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Scheduler client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of scheduler client model. */
   err = RM_MESH_SCHEDULER_CLT_GetModelHandle(&g_mesh_scheduler_clt0_ctrl,
&model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_SCHEDULER_CLT_SendReliablePdu()

fsp_err_t RM_MESH_SCHEDULER_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_SCHEDULER_CLT_SendReliablePdu(&g_mesh_scheduler_clt0_ctrl,
req_opcode, p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCHEDULER_CLT_Get()

fsp_err_t RM_MESH_SCHEDULER_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Scheduler Get is an acknowledged message used to get the current Schedule Register state of an element. The response to the Scheduler Get message is a Scheduler Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scheduler_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCHEDULER_CLT_ActionGet()

fsp_err_t RM_MESH_SCHEDULER_CLT_ActionGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scheduler Action Get is an acknowledged message used to report the action defined by the entry of the Schedule Register state of an element, identified by the Index field. The response to the Scheduler Action Get message is a Scheduler Action Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scheduler_clt control block. |
| [in] | p_parameter | Pointer to Scheduler Action Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCHEDULER_CLT_ActionSet()

fsp_err_t RM_MESH_SCHEDULER_CLT_ActionSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scheduler Action Set is an acknowledged message used to set the entry of the Schedule Register state of an element, identified by the Index field. The response to the Scheduler Action Set message is a Scheduler Action Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scheduler_clt control block. |
| [in] | p_parameter | Pointer to Scheduler Action Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SCHEDULER_CLT_ActionSetUnacknowledged()

fsp_err_t RM_MESH_SCHEDULER_CLT_ActionSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Scheduler Action Set Unacknowledged is an unacknowledged message used to set the entry of the Schedule Register state of an element, identified by the Index field.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_scheduler_clt control block. |
| [in] | p_parameter | Pointer to Scheduler Action Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Scheduler Server (rm_mesh_scheduler_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SCHEDULER_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SCHEDULER_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SCHEDULER_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Scheduler Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_scheduler_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_scheduler_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Scheduler Server (rm_mesh_scheduler_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Scheduler Server (rm_mesh_scheduler_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scheduler server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh scheduler server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_scheduler_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_scheduler_srv_schedules_info_t |
| struct | rm_mesh_scheduler_srv_entry_index_info_t |
| struct | rm_mesh_scheduler_srv_entry_info_t |
| struct | rm_mesh_scheduler_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_scheduler_srv_schedules_info_t

| struct rm_mesh_scheduler_srv_schedules_info_t | | |
|---|---|---|
| The current Schedule Register state of an element. | | |
| Data Fields | | |
| uint16_t | schedules | Bit field indicating defined Actions in the Schedule Register |

### ◆ rm_mesh_scheduler_srv_entry_index_info_t

| struct rm_mesh_scheduler_srv_entry_index_info_t | | |
|---|---|---|
| The entry index of the Schedule Register state | | |
| Data Fields | | |
| uint8_t | index | Index of the Schedule Register entry |

### ◆ rm_mesh_scheduler_srv_entry_info_t

| struct rm_mesh_scheduler_srv_entry_info_t | | |
|---|---|---|
| The entry of the Schedule Register state | | |
| Data Fields | | |
| uint8_t | index | Index of the Schedule Register entry |
| uint8_t | year | Scheduled year for the action |
| uint16_t | month | Scheduled month for the action |
| uint8_t | day | Scheduled day of the month for the action |
| uint8_t | hour | Scheduled hour for the action |
| uint8_t | minute | Scheduled minute for the action |
| uint8_t | second | Scheduled second for the action |
| uint8_t | dayofweek | Schedule days of the week for the action |
| uint8_t | action | Action to be performed at the scheduled time |
| uint8_t | transition_time | Transition time for this action |
| uint16_t | scene_number | Scene number to be used for some actions |

### ◆ rm_mesh_scheduler_srv_instance_ctrl_t

| struct rm_mesh_scheduler_srv_instance_ctrl_t |
|---|
| BLE mesh scheduler instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SCHEDULER_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_SCHEDULER_SRV_Open()

| fsp_err_t RM_MESH_SCHEDULER_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* ) |
|---|
| API to initialize Scheduler Server model. This is to initialize Scheduler Server model and to register with Access layer.<br>Implements rm_ble_mesh_model_server_api_t::open.<br>Example:<br>`/* Open the module. */`<br>`err = RM_MESH_SCHEDULER_SRV_Open(&g_mesh_scheduler_srv0_ctrl, &g_mesh_scheduler_srv0_cfg);`<br>**Return values**<br>FSP_SUCCESS — Model opened successfully.<br>FSP_ERR_ASSERTION — Pointer to control block or configuration structure is NULL.<br>FSP_ERR_ALREADY_OPEN — Model is already open.<br>FSP_ERR_NOT_FOUND — The number of models has exceeded the limit.<br>FSP_ERR_ABORTED — Model initialization is failed. |

### ◆ RM_MESH_SCHEDULER_SRV_Close()

fsp_err_t RM_MESH_SCHEDULER_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Scheduler Server model. This is to terminate Scheduler Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SCHEDULER_SRV_Close(&g_mesh_scheduler_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SCHEDULER_SRV_StateUpdate()

fsp_err_t RM_MESH_SCHEDULER_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_SCHEDULER_SRV_StateUpdate(&g_mesh_scheduler_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Sensor Client (rm_mesh_sensor_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SENSOR_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SENSOR_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SENSOR_CLT_GetModelHandle (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_SENSOR_CLT_DescriptorGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_ColumnGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SeriesGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_CadenceGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_CadenceSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_CadenceSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SettingsGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SettingGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SettingSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_SENSOR_CLT_SettingSetUnacknowledged (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Sensor Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_sensor_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_sensor_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Sensor Client (rm_mesh_sensor_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Sensor Client (rm_mesh_sensor_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh sensor client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_sensor_clt0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_sensor_clt_instance_ctrl_t |

### Data Structure Documentation

#### ◆ rm_mesh_sensor_clt_instance_ctrl_t

| struct rm_mesh_sensor_clt_instance_ctrl_t |
|---|
| BLE mesh sensor instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SENSOR_CLT_Open() is called. |

### Function Documentation

### ◆ RM_MESH_SENSOR_CLT_Open()

fsp_err_t RM_MESH_SENSOR_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Sensor Client middleware. This is to initialize Sensor Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_SENSOR_CLT_Open(&g_mesh_sensor_clt0_ctrl, &g_mesh_sensor_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_SENSOR_CLT_Close()

fsp_err_t RM_MESH_SENSOR_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Sensor Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SENSOR_CLT_Close(&g_mesh_sensor_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SENSOR_CLT_GetModelHandle()

fsp_err_t RM_MESH_SENSOR_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Sensor client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of sensor client model. */
   err = RM_MESH_SENSOR_CLT_GetModelHandle(&g_mesh_sensor_clt0_ctrl, &model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_SENSOR_CLT_SendReliablePdu()

fsp_err_t RM_MESH_SENSOR_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_SENSOR_CLT_SendReliablePdu(&g_mesh_sensor_clt0_ctrl, req_opcode,
p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_DescriptorGet()

fsp_err_t RM_MESH_SENSOR_CLT_DescriptorGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Descriptor Get is an acknowledged message used to get the Sensor Descriptor state of all sensors within an element. The response to a Sensor Descriptor Get message is a Sensor Descriptor Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Descriptor Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_Get()

fsp_err_t RM_MESH_SENSOR_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Get is an acknowledged message used to get the Sensor Data state. The response to the Sensor Get message is a Sensor Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_ColumnGet()

fsp_err_t RM_MESH_SENSOR_CLT_ColumnGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Column Get is an acknowledged message used to get the Sensor Series Column state. The response to the Sensor Column Get message is a Sensor Column Status message

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Column Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_SeriesGet()

fsp_err_t RM_MESH_SENSOR_CLT_SeriesGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Series Get is an acknowledged message used to get a sequence of the Sensor Series Column states. The response to the Sensor Series Get message is a Sensor Series Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Series Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_CadenceGet()

fsp_err_t RM_MESH_SENSOR_CLT_CadenceGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Cadence Get is an acknowledged message used to get the Sensor Cadence state of an element. The response to the Sensor Cadence Get message is a Sensor Cadence Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Cadence Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_CadenceSet()

fsp_err_t RM_MESH_SENSOR_CLT_CadenceSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Cadence Set is an acknowledged message used to set the Sensor Cadence state of an element. The response to the Sensor Cadence Set message is a Sensor Cadence Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Cadence Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_CadenceSetUnacknowledged()

fsp_err_t RM_MESH_SENSOR_CLT_CadenceSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Cadence Set Unacknowledged is an unacknowledged message used to set the Sensor Cadence state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Cadence Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_SettingsGet()

fsp_err_t RM_MESH_SENSOR_CLT_SettingsGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Settings Get is an acknowledged message used to get the list of Sensor Setting states of an element. The response to the Sensor Settings Get message is a Sensor Settings Status message

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Settings Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_SettingGet()

fsp_err_t RM_MESH_SENSOR_CLT_SettingGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Setting Get is an acknowledged message used to get the Sensor Setting state of an element. The response to the Sensor Setting Get message is a Sensor Setting Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Settings Get message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_SettingSet()

fsp_err_t RM_MESH_SENSOR_CLT_SettingSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Setting Set is an acknowledged message used to set the Sensor Setting state of an element. The response to the Sensor Setting Set message is a Sensor Setting Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Settings Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_CLT_SettingSetUnacknowledged()

fsp_err_t RM_MESH_SENSOR_CLT_SettingSetUnacknowledged ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Sensor Setting Set Unacknowledged is an unacknowledged message used to set the Sensor Setting state of an element.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_clt control block. |
| [in] | p_parameter | Pointer to Sensor Settings Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Sensor Server (rm_mesh_sensor_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_SENSOR_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_SENSOR_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_SENSOR_SRV_StateUpdate |

| | |
|---|---|
| | (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |
| fsp_err_t | RM_MESH_SENSOR_SRV_SetupServerStateUpdate (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Sensor Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_sensor_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_sensor_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Sensor Server (rm_mesh_sensor_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Sensor Server (rm_mesh_sensor_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh sensor server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh sensor server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_sensor_srv0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_sensor_srv_descriptor_info_t |
| struct | rm_mesh_sensor_srv_settings_info_t |
| | |

| | |
|---|---|
| struct | rm_mesh_sensor_srv_setting_info_t |
| struct | rm_mesh_sensor_srv_cadence_info_t |
| struct | rm_mesh_sensor_srv_data_info_t |
| struct | rm_mesh_sensor_srv_series_column_info_t |
| struct | rm_mesh_sensor_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_sensor_srv_descriptor_info_t

struct rm_mesh_sensor_srv_descriptor_info_t

Sensor Descriptor state represents the attributes describing the sensor data

Data Fields

| | | |
|---|---|---|
| uint16_t | sensor_property_id | Sensor Property ID field is a 2-octet value referencing a device property that describes the meaning and the format of data reported by a sensor |
| uint16_t | sensor_positive_tolerance | Sensor Positive Tolerance field is a 12-bit value representing the magnitude of a possible positive error associated with the measurements that the sensor is reporting |
| uint16_t | sensor_negative_tolerance | Sensor Negative Tolerance field is a 12-bit value representing the magnitude of a possible negative error associated with the measurements that the sensor is reporting |
| uint8_t | sensor_sampling_function | Sensor Sampling Function field specifies the averaging operation or type of sampling function applied to the measured value |
| uint8_t | sensor_measurement_period | Sensor Measurement Period field specifies a uint8 value n that represents the averaging time span, accumulation time, or measurement period in seconds over which the measurement is taken |
| uint8_t | sensor_update_interval | measurement reported by a sensor is internally refreshed at |

| | | |
|---|---|---|
| | | the frequency indicated in the Sensor Update Interval field |
| uint8_t | status | Status - used in response to indicate if other fields to be included |

### ◆ rm_mesh_sensor_srv_settings_info_t

| struct rm_mesh_sensor_srv_settings_info_t | | |
|---|---|---|
| Sensor Settings state controls parameters of a sensor | | |
| Data Fields | | |
| uint16_t | sensor_property_id | Property ID of a sensor |
| uint16_t * | setting_property_ids | Property ID of a setting within a sensor |
| uint16_t | setting_property_ids_count | |

### ◆ rm_mesh_sensor_srv_setting_info_t

| struct rm_mesh_sensor_srv_setting_info_t | | |
|---|---|---|
| Sensor Setting state controls parameters of a sensor | | |
| Data Fields | | |
| uint16_t | sensor_property_id | Property ID of a sensor |
| uint16_t | sensor_setting_property_id | Property ID of a setting within a sensor |
| uint8_t | sensor_setting_access | Read/Write access rights for the setting |
| uint8_t * | sensor_setting_raw | Raw value of a setting within a sensor |
| uint16_t | sensor_setting_raw_len | |
| uint8_t | status | |

### ◆ rm_mesh_sensor_srv_cadence_info_t

| struct rm_mesh_sensor_srv_cadence_info_t | | |
|---|---|---|
| Sensor Cadence state controls the cadence of sensor reports | | |
| Data Fields | | |
| uint16_t | sensor_property_id | Property ID of a sensor |
| uint8_t | fast_cadence_period_divisor | Divisor for the Publish Period |
| uint8_t | status_trigger_type | Defines the unit and format of the Status Trigger Delta fields |
| uint8_t * | status_trigger_delta_down | Delta down value that triggers a status message |
| uint16_t | status_trigger_delta_down_len | |

| | | |
|---|---|---|
| uint8_t * | status_trigger_delta_up | Delta up value that triggers a status message |
| uint16_t | status_trigger_delta_up_len | |
| uint8_t | status_min_interval | Minimum interval between two consecutive Status messages APPLICATION NOTE: The Current Sensor Server Model implementation does not inherently check for the time interval between two consecutive status messages. The application layer which manages the data for the Sensor Server Model holds the responsibility for interleaving consecutive status messages with the configured Minimum time interval for statuses. |
| uint8_t * | fast_cadence_low | Low value for the fast cadence range |
| uint16_t | fast_cadence_low_len | |
| uint8_t * | fast_cadence_high | High value for the fast cadence range |
| uint16_t | fast_cadence_high_len | |
| uint8_t | status | Status - used in response path |

### ◆ rm_mesh_sensor_srv_data_info_t

| struct rm_mesh_sensor_srv_data_info_t | | |
|---|---|---|
| The Sensor Data state is a sequence of one or more pairs of Sensor Property ID and Raw Value fields, with each Raw Value field size and representation defined by the characteristics referenced by the Sensor Property ID | | |
| Data Fields | | |
| uint16_t | property_id_1 | ID of the 1st device property of the sensor |
| uint8_t * | raw_value_1 | Raw Value field with a size and representation defined by the 1st device property |
| uint16_t | raw_value_1_len | |
| uint16_t | property_id_2 | ID of the 2nd device property of the sensor |
| uint8_t * | raw_value_2 | Raw Value field with a size and representation defined by the 2nd device property |
| uint16_t | raw_value_2_len | |

| | | |
|---|---|---|
| uint16_t | property_id_n | ID of the nth device property of the sensor |
| uint8_t * | raw_value_n | Raw Value field with a size and representation defined by the nth device property |
| uint16_t | raw_value_n_len | |
| uint8_t | status | Status - used in response path |

### ◆ rm_mesh_sensor_srv_series_column_info_t

| struct rm_mesh_sensor_srv_series_column_info_t | | |
|---|---|---|
| Values measured by sensors may be organized as arrays (and represented as series of columns, such as histograms | | |
| Data Fields | | |
| uint16_t | sensor_property_id | Property describing the data series of the sensor |
| uint8_t * | sensor_raw_value_x | Raw value representing the left corner of a column on the X axis |
| uint16_t | sensor_raw_value_x_len | |
| uint8_t * | sensor_column_width | Raw value representing the width of the column |
| uint16_t | sensor_column_width_len | |
| uint8_t * | sensor_raw_value_y | Raw value representing the height of the column on the Y axis |
| uint16_t | sensor_raw_value_y_len | |
| uint8_t | status | Status - used in response path |

### ◆ rm_mesh_sensor_srv_instance_ctrl_t

| struct rm_mesh_sensor_srv_instance_ctrl_t |
|---|
| BLE mesh sensor instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_SENSOR_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_SENSOR_SRV_Open()

fsp_err_t RM_MESH_SENSOR_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Sensor Server model and to initialize Sensor_Setup Server model. This is to initialize Sensor Server model and to register with Access layer. And this is to initialize Sensor_Setup Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_SENSOR_SRV_Open(&g_mesh_sensor_srv0_ctrl, &g_mesh_sensor_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_SENSOR_SRV_Close()

fsp_err_t RM_MESH_SENSOR_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Sensor Server model. This is to terminate Sensor Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_SENSOR_SRV_Close(&g_mesh_sensor_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_SENSOR_SRV_StateUpdate()

fsp_err_t RM_MESH_SENSOR_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_SENSOR_SRV_StateUpdate(&g_mesh_sensor_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_SENSOR_SRV_SetupServerStateUpdate()

fsp_err_t RM_MESH_SENSOR_SRV_SetupServerStateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_sensor_srv control block. |
| [in] | p_state | To send reply for a request or to inform change in state. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Time Client (rm_mesh_time_clt)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_TIME_CLT_Open (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_TIME_CLT_Close (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_CLT_GetModelHandle |

| | |
|---|---|
| | (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| fsp_err_t | RM_MESH_TIME_CLT_SendReliablePdu (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| fsp_err_t | RM_MESH_TIME_CLT_Get (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_CLT_Set (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_TIME_CLT_ZoneGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_CLT_ZoneSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_TIME_CLT_TaiUtcDeltaGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_CLT_TaiUtcDeltaSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |
| fsp_err_t | RM_MESH_TIME_CLT_RoleGet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_CLT_RoleSet (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, void const *const p_parameter) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Time Client module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_time_clt

The following build time configurations are defined in fsp_cfg/rm_mesh_time_clt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Time Client (rm_mesh_time_clt)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Time Client (rm_mesh_time_clt).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh time client ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_time_clt0 | Module name. |

**Data Structures**

| | |
|---|---|
| struct | rm_mesh_time_clt_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_time_clt_instance_ctrl_t

| struct rm_mesh_time_clt_instance_ctrl_t |
|---|
| BLE mesh time instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_TIME_CLT_Open() is called. |

## Function Documentation

### ◆ RM_MESH_TIME_CLT_Open()

fsp_err_t RM_MESH_TIME_CLT_Open ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_client_cfg_t const *const *p_cfg* )

Open Time Client middleware. This is to initialize Time Client model and to register with Access layer.

Implements rm_ble_mesh_model_client_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_TIME_CLT_Open(&g_mesh_time_clt0_ctrl, &g_mesh_time_clt0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_TIME_CLT_Close()

fsp_err_t RM_MESH_TIME_CLT_Close ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Close Time Client middleware.

Implements rm_ble_mesh_model_client_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_TIME_CLT_Close(&g_mesh_time_clt0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_TIME_CLT_GetModelHandle()

fsp_err_t RM_MESH_TIME_CLT_GetModelHandle ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_model_handle_t *const *p_model_handle* )

This is to get the handle of Time client model.

Implements rm_ble_mesh_model_client_api_t::getModelHandle.

Example:

```
/* Get the handle of time client model. */
   err = RM_MESH_TIME_CLT_GetModelHandle(&g_mesh_time_clt0_ctrl, &model_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Got model handle successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_POINTER | The parameter p_model_handle is NULL. |

### ◆ RM_MESH_TIME_CLT_SendReliablePdu()

fsp_err_t RM_MESH_TIME_CLT_SendReliablePdu ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, uint32_t *req_opcode*, void const *const *p_parameter*, uint32_t *rsp_opcode* )

This is to initialize sending acknowledged commands.

Implements rm_ble_mesh_model_client_api_t::sendReliablePdu.

Example:

```
/* Initialize sending acknowledged commands. */
   err = RM_MESH_TIME_CLT_SendReliablePdu(&g_mesh_time_clt0_ctrl, req_opcode,
p_parameter, rsp_opcode);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_Get()

fsp_err_t RM_MESH_TIME_CLT_Get ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Time Get is a message used to get the Time state of neighbor nodes. The response to the Time Get message is a Time Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_Set()

fsp_err_t RM_MESH_TIME_CLT_Set ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Time Set is an acknowledged message used to set the Time state of an element. The response to the Time Set message is a Time Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |
| [in] | p_parameter | Pointer to Time Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_ZoneGet()

fsp_err_t RM_MESH_TIME_CLT_ZoneGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Time Zone Get is an acknowledged message used to get the Time Zone Offset Current state, the Time Zone Offset New state , and the TAI of Zone Change state. The response to the Time Zone Get message is a Time Zone Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_ZoneSet()

fsp_err_t RM_MESH_TIME_CLT_ZoneSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Time Zone Set is an acknowledged message used to set the Time Zone Offset New state and the TAI of Zone Change state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |
| [in] | p_parameter | Pointer to Time Zone Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_TaiUtcDeltaGet()

fsp_err_t RM_MESH_TIME_CLT_TaiUtcDeltaGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

TAI-UTC Delta Get is an acknowledged message used to get the TAI-UTC Delta Current state, the TAI-UTC Delta New state, and the TAI of Delta Change state. The response to the TAI-UTC Delta Get message is a TAI-UTC Delta Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_TaiUtcDeltaSet()

fsp_err_t RM_MESH_TIME_CLT_TaiUtcDeltaSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

TAI-UTC Delta Set is an acknowledged message used to set the TAI-UTC Delta New state and the TAI of Delta Change state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |
| [in] | p_parameter | Pointer to TAI-UTC Delta Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_RoleGet()

fsp_err_t RM_MESH_TIME_CLT_RoleGet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*)

Time Role Get is an acknowledged message used to get the Time Role state of an element. The response to the Time Role Get message is a Time Role Status message. There are no parameters for this message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

### ◆ RM_MESH_TIME_CLT_RoleSet()

fsp_err_t RM_MESH_TIME_CLT_RoleSet ( rm_ble_mesh_model_client_ctrl_t *const *p_ctrl*, void const *const *p_parameter* )

Time Role Set is an acknowledged message used to set the Time Role state of an element. The response to the Time Role Set message is a Time Role Status message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | rm_mesh_time_clt control block. |
| [in] | p_parameter | Pointer to Time Role Set message. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_INVALID_POINTER | The parameter p_parameter is NULL. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Time Server (rm_mesh_time_srv)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_MESH_TIME_SRV_Open (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MESH_TIME_SRV_Close (rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_MESH_TIME_SRV_StateUpdate (rm_ble_mesh_model_server_ctrl_t |

| | |
|---|---|
| | *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

### Detailed Description

# Overview

### Target Devices

The BLE Mesh Network Time Server module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_mesh_time_srv

The following build time configurations are defined in fsp_cfg/rm_mesh_time_srv_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Model Time Server (rm_mesh_time_srv)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Model Time Server (rm_mesh_time_srv).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Callback Provided when Timeout Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh time server ISR occurs |
| Callback Provided when an Timeout ISR Occurs | Name Must Be a Valid C Symbol | NULL | Callback provided when mesh time server timeout ISR occurs |
| Name | Name Must Be a Valid C Symbol | g_rm_mesh_time_srv0 | Module name. |

### Data Structures

| | |
|---|---|
| struct | rm_mesh_time_srv_info_t |
| struct | rm_mesh_time_srv_zone_info_t |
| struct | rm_mesh_time_srv_tai_utc_delta_info_t |
| struct | rm_mesh_time_srv_role_info_t |
| struct | rm_mesh_time_srv_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_mesh_time_srv_info_t

| struct rm_mesh_time_srv_info_t | | |
|---|---|---|
| Mesh defines times based on International Atomic Time (TAI). The base representation of times is the number of seconds after 00:00:00 TAI on 2000-01-01 (that is, 1999-12-31T23:59:28 UTC) | | |
| Data Fields | | |
| uint8_t | tai_seconds[5] | Current TAI time in seconds since the epoch. |
| uint8_t | subsecond | The sub-second time in units of 1/256s. |
| uint8_t | uncertainty | Estimated uncertainty in 10-millisecond steps. |
| uint8_t | time_authority | 0 = No Time Authority. The element does not have a trusted OOB source of time, such as GPS or NTP. 1 = Time Authority. The element has a trusted OOB source of time, such as GPS or NTP or a battery-backed, properly initialized RTC. |
| uint16_t | tai_utc_delta | Current difference between TAI and UTC in seconds |
| uint8_t | time_zone_offset | The local time zone offset in 15-minute increments |

### ◆ rm_mesh_time_srv_zone_info_t

| struct rm_mesh_time_srv_zone_info_t | | |
|---|---|---|
| Time Zone | | |
| Data Fields | | |
| uint8_t | time_zone_offset_current | Current local time zone offset. Meaningful only in 'Time Zone Status' response. |
| uint8_t | time_zone_offset_new | Upcoming local time zone offset. |
| uint8_t | tai_of_zone_change[5] | Absolute TAI time when the Time Zone Offset will change from Current to New. |

### ◆ rm_mesh_time_srv_tai_utc_delta_info_t

| struct rm_mesh_time_srv_tai_utc_delta_info_t |
|---|
| TAI-UTC Delta |

| Data Fields | | |
|---|---|---|
| uint16_t | tai_utc_delta_current | Current difference between TAI and UTC in seconds. Meaningful only in 'TAI-UTC Delta Status' response. |
| uint16_t | tai_utc_delta_new | Upcoming difference between TAI and UTC in seconds. |
| uint8_t | padding | Always 0b0. Other values are Prohibited. |
| uint8_t | tai_of_delta_change[5] | TAI Seconds time of the upcoming TAI-UTC Delta change |

### ◆ rm_mesh_time_srv_role_info_t

| struct rm_mesh_time_srv_role_info_t | | |
|---|---|---|
| The Time Role state of an element | | |
| Data Fields | | |
| uint8_t | role | Time Role |

### ◆ rm_mesh_time_srv_instance_ctrl_t

| struct rm_mesh_time_srv_instance_ctrl_t |
|---|
| BLE mesh time instance control block. DO NOT INITIALIZE. Initialization occurs when RM_MESH_TIME_SRV_Open() is called. |

## Function Documentation

### ◆ RM_MESH_TIME_SRV_Open()

fsp_err_t RM_MESH_TIME_SRV_Open ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_model_server_cfg_t const *const *p_cfg* )

API to initialize Time Server model. This is to initialize Time Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::open.

Example:

```
/* Open the module. */
   err = RM_MESH_TIME_SRV_Open(&g_mesh_time_srv0_ctrl, &g_mesh_time_srv0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Model is already open. |
| FSP_ERR_NOT_FOUND | The number of models has exceeded the limit. |
| FSP_ERR_ABORTED | Model initialization is failed. |

### ◆ RM_MESH_TIME_SRV_Close()

fsp_err_t RM_MESH_TIME_SRV_Close ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*)

API to terminate Time Server model. This is to terminate Time Server model and to register with Access layer.

Implements rm_ble_mesh_model_server_api_t::close.

Example:

```
/* Close the module. */
   err = RM_MESH_TIME_SRV_Close(&g_mesh_time_srv0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Model successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |

### ◆ RM_MESH_TIME_SRV_StateUpdate()

fsp_err_t RM_MESH_TIME_SRV_StateUpdate ( rm_ble_mesh_model_server_ctrl_t *const *p_ctrl*, rm_ble_mesh_access_server_state_t const *const *p_state* )

API to send reply or to update state change. This is to send reply for a request or to inform change in state.

Implements rm_ble_mesh_model_server_api_t::stateUpdate.

Example:

```
/* Update server status. */
   err = RM_MESH_TIME_SRV_StateUpdate(&g_mesh_time_srv0_ctrl, &state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Updated server status successfully. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_state is NULL. |
| FSP_ERR_NOT_OPEN | Model is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_APPROXIMATION | Lower layer is invalid state. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_ABORTED | Operation is failed. |

## BLE Mesh Network Upper Trans (rm_ble_mesh_upper_trans)

Modules » Networking » BLE Mesh Network Modules

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_Open (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_Close (rm_ble_mesh_provision_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_SendAccessPdu |

| | |
|---|---|
| | (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_access_layer_pdu_t const *const p_access_layer_pdu, rm_ble_mesh_lower_trans_reliable_t reliable) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_SendControlPdu (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_pdu_t const *const p_control_pdu) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_LpnSetupFriendship (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friendship_setting_t const *const p_setting) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_LpnClearFriendship (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_LpnManageSubscription (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_opcode_t action, uint16_t const *const p_addr_list, uint16_t count) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_LpnPoll (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_IsValidLpnElementAddress (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_IsValidLpnSubscriptionAddress (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_GetLpnPolltimeout (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t lpn_addr, uint32_t *const p_poll_timeout) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_GetFriendshipInfo (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friend_role_t role, uint16_t lpn_index, rm_ble_mesh_upper_trans_friendship_info_t *const p_node) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_LpnRegisterSecurityUpdate (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t flag, uint32_t ivindex) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_ClearAllLpn (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_SetHeartbeatPublication (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_GetHeartbeatPublication (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_SetHeartbeatSubscription (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_GetHeartbeatSubscription (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info) |
| fsp_err_t | RM_BLE_MESH_UPPER_TRANS_TriggerHeartbeat (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, uint8_t change_in_feature_bit) |

**Detailed Description**

# Overview

### Target Devices

The BLE Mesh Network Upper Trans module supports the following devices.

- RA4W1

# Configuration

### Build Time Configurations for rm_ble_mesh_upper_trans

The following build time configurations are defined in fsp_cfg/rm_ble_mesh_upper_trans_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

### Configurations for Networking > BLE Mesh Network modules > BLE Mesh Upper Trans (rm_ble_mesh_upper_trans)

This module can be added to the Stacks tab via New Stack > Networking > BLE Mesh Network modules > BLE Mesh Upper Trans (rm_ble_mesh_upper_trans).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Name | Name Must Be a Valid C Symbol | g_rm_ble_mesh_upper_trans0 | Module name. |
| Channel Number | Invalid Channel Number | 0 | Select channel corresponding to the channel number of the hardware. |
| Control Message Event Enable | • Enabled<br>• Disabled | Disabled | Whether to enable the control message event or not. |
| Access Message Event Enable | • Enabled<br>• Disabled | Disabled | Whether to enable the access message event or not. |
| Callback | Name Must Be a Valid C Symbol | NULL | Callback function name. |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_upper_trans_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_upper_trans_instance_ctrl_t

| struct rm_ble_mesh_upper_trans_instance_ctrl_t |
|---|
| RM_BLE_MESH_UPPER_TRANS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_MESH_UPPER_TRANS_Open() is called. |

## Function Documentation

### ◆ RM_BLE_MESH_UPPER_TRANS_Open()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_Open ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_cfg_t const *const *p_cfg* )

Register Inerface with Transport Layer. This routine registers interface with the Transport Layer. Transport Layer supports single Application, hence this rouine shall be called once.

Implements rm_ble_mesh_upper_trans_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_MESH_UPPER_TRANS_Open(&g_ble_mesh_upper_trans0_ctrl,
&g_ble_mesh_upper_trans0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_Close()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_Close ( rm_ble_mesh_provision_ctrl_t *const *p_ctrl*)

Unregister Inerface with Transport Layer. Implements rm_ble_mesh_upper_trans_api_t::close.

Example:

```
/* Close the module. */
   err = RM_BLE_MESH_UPPER_TRANS_Close(&g_ble_mesh_upper_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module successfully closed. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_UPPER_TRANS_SendAccessPdu()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_SendAccessPdu ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_access_layer_pdu_t const *const *p_access_layer_pdu*, rm_ble_mesh_lower_trans_reliable_t *reliable* )

API to send Access Layer PDUs. This routine sends Access Layer PDUs to peer device.

Implements rm_ble_mesh_upper_trans_api_t::sendAccessPdu.

Example:

```
/* Send Access Layer PDUs. */
   err = RM_BLE_MESH_UPPER_TRANS_SendAccessPdu(&g_ble_mesh_upper_trans0_ctrl,
                                               &access_layer_pdu,
RM_BLE_MESH_LOWER_TRANS_RELIABLE_ENABLE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_access_layer_pdu is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_UPPER_TRANS_SendControlPdu()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_SendControlPdu ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_control_pdu_t const *const *p_control_pdu* )

API to send transport Control PDUs. This routine sends transport Control PDUs to peer device.

Implements rm_ble_mesh_upper_trans_api_t::sendControlPdu.

Example:

```
/* Send Upper Transport control PDUs. */
   err = RM_BLE_MESH_UPPER_TRANS_SendControlPdu(&g_ble_mesh_upper_trans0_ctrl,
&control_pdu);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_control_pdu is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_UPPER_TRANS_LpnSetupFriendship()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_LpnSetupFriendship ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_friendship_setting_t const *const *p_setting* )

API to setup Friendship. This routine is used by the device acting as a low power node to setup a friendship procedure to any available friend nodes.

Implements rm_ble_mesh_upper_trans_api_t::lpnSetupFriendship.

Example:

```
/* Setup friendship. */
   err = RM_BLE_MESH_UPPER_TRANS_LpnSetupFriendship(&g_ble_mesh_upper_trans0_ctrl,
&setting);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_setting is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Lower layer is invalid state. |

### ◆ RM_BLE_MESH_UPPER_TRANS_LpnClearFriendship()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_LpnClearFriendship ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*)

API to terminate friendship. This routine is used by the device acting as a low power node terminate friendship with an active Friend node.

Implements rm_ble_mesh_upper_trans_api_t::lpnClearFriendship.

Example:

```
/* Terminate friendship. */
   err = RM_BLE_MESH_UPPER_TRANS_LpnClearFriendship(&g_ble_mesh_upper_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_LpnManageSubscription()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_LpnManageSubscription ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_control_opcode_t *action*, uint16_t const *const *p_addr_list*, uint16_t *count* )

API to manage friend subscription list. This routine is used by the device acting as a low power node add/remove addresses to/from the friends subscription list.

Implements rm_ble_mesh_upper_trans_api_t::lpnManageSubscription.

Example:

```
/* Manage friend subscription list. */
   err =
RM_BLE_MESH_UPPER_TRANS_LpnManageSubscription(&g_ble_mesh_upper_trans0_ctrl,
 RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_POLL,
                                                &addr_list,
                                                count);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_addr_list is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |

### ◆ RM_BLE_MESH_UPPER_TRANS_LpnPoll()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_LpnPoll ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*)

To trigger Friend Poll from application. This routine enables the application to trigger a Friend Poll even before the expiry of an active poll period configured during the friendship establishment. The poll peroid will get reset at this point.

Implements rm_ble_mesh_upper_trans_api_t::lpnPoll.

Example:

```
/* Trigger friend poll from application. */
   err = RM_BLE_MESH_UPPER_TRANS_LpnPoll(&g_ble_mesh_upper_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_IsValidLpnElementAddress()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_IsValidLpnElementAddress ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr*, rm_ble_mesh_lower_trans_lpn_handle_t *const *p_lpn_handle* )

To check if address matches with any of the LPN. This routine checks if destination address in a received packet matches with any of the known element address of LPN.

Implements rm_ble_mesh_upper_trans_api_t::isValidLpnElementAddress.

Example:

```
/* Check if address matches with any of the LPN. */
   err =
RM_BLE_MESH_UPPER_TRANS_IsValidLpnElementAddress(&g_ble_mesh_upper_trans0_ctrl, addr,
&lpn_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_lpn_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_UPPER_TRANS_IsValidLpnSubscriptionAddress()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_IsValidLpnSubscriptionAddress ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *addr*, rm_ble_mesh_lower_trans_lpn_handle_t *const *p_lpn_handle* )

To check if valid subscription address of an LPN to receive a packet. This routine checks if destination address in a received packet matches with any of the known subscription address of an LPN.

Implements rm_ble_mesh_upper_trans_api_t::isValidLpnSubscriptionAddress.

Example:

```
/* Check if valid subscription address of an LPN to receive a packet. */
   err =
RM_BLE_MESH_UPPER_TRANS_IsValidLpnSubscriptionAddress(&g_ble_mesh_upper_trans0_ctrl,
addr, &lpn_handle);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_lpn_handle is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |

### ◆ RM_BLE_MESH_UPPER_TRANS_GetLpnPolltimeout()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_GetLpnPolltimeout ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_address_t *lpn_addr*, uint32_t *const *p_poll_timeout* )

To get Poll Timeout of an LPN. This routine checks if LPN address is valid and then returns Poll Timeout configured for the LPN.

Implements rm_ble_mesh_upper_trans_api_t::getLpnPolltimeout.

Example:

```
/* Get poll timeout of an LPN. */
   err = RM_BLE_MESH_UPPER_TRANS_GetLpnPolltimeout(&g_ble_mesh_upper_trans0_ctrl,
addr, &poll_timeout);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_poll_timeout is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_GetFriendshipInfo()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_GetFriendshipInfo ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_friend_role_t *role*, uint16_t *lpn_index*, rm_ble_mesh_upper_trans_friendship_info_t *const *p_node* )

To get the LPN node information. This routine fetches the node information of the LPN element at the given index

Implements rm_ble_mesh_upper_trans_api_t::getFriendshipInfo.

Example:

```
/* Get the LPN node information. */
   err = RM_BLE_MESH_UPPER_TRANS_GetFriendshipInfo(&g_ble_mesh_upper_trans0_ctrl,
RM_BLE_MESH_UPPER_TRANS_FRIEND_ROLE_FRIEND,
                                                   lpn_index,
                                                   &node);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_node is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_LpnRegisterSecurityUpdate()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_LpnRegisterSecurityUpdate ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_network_subnet_handle_t *subnet_handle*, uint8_t *flag*, uint32_t *ivindex* )

To add the security update information. This routine updates the security state of the network to all the active LPN elements. This will be forwarded to the elements when it polls for the next packet available.

Implements rm_ble_mesh_upper_trans_api_t::lpnRegisterSecurityUpdate.

Example:

```
/* Add the security update information. */
   err =
RM_BLE_MESH_UPPER_TRANS_LpnRegisterSecurityUpdate(&g_ble_mesh_upper_trans0_ctrl,
subnet_handle, flag, ivindex);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_ClearAllLpn()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_ClearAllLpn ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*)

To clear information related to all LPNs. This routine clears information related to all LPNs.

Implements rm_ble_mesh_upper_trans_api_t::clearAllLpn.

Example:

```
/* Clear information related to all LPNs. */
   err = RM_BLE_MESH_UPPER_TRANS_ClearAllLpn(&g_ble_mesh_upper_trans0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_SetHeartbeatPublication()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_SetHeartbeatPublication ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const *p_info* )

To set the Heartbeat publication data. This routine configures the Heartbeat publication information

Implements rm_ble_mesh_upper_trans_api_t::setHeartbeatPublication.

Example:

```
 /* Set the heartbeat publication data. */
    err =
RM_BLE_MESH_UPPER_TRANS_SetHeartbeatPublication(&g_ble_mesh_upper_trans0_ctrl,
&ut_hp_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |

### ◆ RM_BLE_MESH_UPPER_TRANS_GetHeartbeatPublication()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_GetHeartbeatPublication ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const *p_info* )

To get the Heartbeat publication data. This routine retrieves the Heartbeat publication information

Implements rm_ble_mesh_upper_trans_api_t::getHeartbeatPublication.

Example:

```
/* Get the heartbeat publication data. */
   err =
RM_BLE_MESH_UPPER_TRANS_GetHeartbeatPublication(&g_ble_mesh_upper_trans0_ctrl,
&ut_hp_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_SetHeartbeatSubscription()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_SetHeartbeatSubscription ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const *p_info* )

To set the Heartbeat subscription data. This routine configures the Heartbeat subscription information

Implements rm_ble_mesh_upper_trans_api_t::setHeartbeatSubscription.

Example:

```
 /* Set the heartbeat subscription data. */
    err =
RM_BLE_MESH_UPPER_TRANS_SetHeartbeatSubscription(&g_ble_mesh_upper_trans0_ctrl,
&ut_hs_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ADDRESS | Invalid source address. |

### ◆ RM_BLE_MESH_UPPER_TRANS_GetHeartbeatSubscription()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_GetHeartbeatSubscription (
rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*,
rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const *p_info* )

To get the Heartbeat subscription data. This routine retrieves the Heartbeat subscription information

Implements rm_ble_mesh_upper_trans_api_t::getHeartbeatSubscription.

Example:

```
/* Get the heartbeat subscription data. */
   err =
RM_BLE_MESH_UPPER_TRANS_GetHeartbeatSubscription(&g_ble_mesh_upper_trans0_ctrl,
&ut_hs_info);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_INVALID_POINTER | The parameter p_info is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLE_MESH_UPPER_TRANS_TriggerHeartbeat()

fsp_err_t RM_BLE_MESH_UPPER_TRANS_TriggerHeartbeat ( rm_ble_mesh_upper_trans_ctrl_t *const *p_ctrl*, uint8_t *change_in_feature_bit* )

To trigger Heartbeat send on change in feature. This routine triggers the Heartbeat send on change in state of supported features.

Implements rm_ble_mesh_upper_trans_api_t::triggerHeartbeat.

Example:

```
/* Trigger heartbeat send on change in feature. */
   err = RM_BLE_MESH_UPPER_TRANS_TriggerHeartbeat(&g_ble_mesh_upper_trans0_ctrl,
change_in_feature_bit);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation succeeded. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input parameter is invalid. |
| FSP_ERR_OUT_OF_MEMORY | Memory allocation is failed. |
| FSP_ERR_OVERFLOW | TX queue is full. |
| FSP_ERR_UNDERFLOW | TX queue is empty. |
| FSP_ERR_NOT_FOUND | Input parameter is not found. |
| FSP_ERR_INVALID_POINTER | The internal parameter is NULL. |

## 5.2.12.12 Cellular Comm Interface on UART (rm_cellular_comm_uart_aws)

Modules » Networking

### Functions

| | |
|---|---|
| CellularCommInterfaceError_t | RM_CELLULAR_COMM_UART_AWS_Open (rm_cellular_comm_uart_aws_instance_ctrl_t *p_instance_ctrl, CellularCommInterfaceReceiveCallback_t receive_callback, void *pUserData) |
| CellularCommInterfaceError_t | RM_CELLULAR_COMM_UART_AWS_Send (CellularCommInterfaceHandle_t commInterfaceHandle, const uint8_t *pData, uint32_t dataLength, uint32_t timeoutMilliseconds, uint32_t *pDataSentLength) |
| CellularCommInterfaceError | RM_CELLULAR_COMM_UART_AWS_Receive |

| | |
|---|---|
| _t | (CellularCommInterfaceHandle_t commInterfaceHandle, uint8_t *pBuffer, uint32_t bufferLength, uint32_t timeoutMilliseconds, uint32_t *pDataReceivedLength) |
| CellularCommInterfaceError_t | RM_CELLULAR_COMM_UART_AWS_Close (CellularCommInterfaceHandle_t commInterfaceHandle) |
| void | comm_uart_aws_callback (uart_callback_args_t *p_args) |

### Detailed Description

Middleware implementing the AWS Cellular Comm Interface for the FSP UART API.

# Overview

See AWS documentation for how the Cellular Comm Interface works: https://www.freertos.org/Documentation/api-ref/cellular/cellular_porting.html#cellular_porting_comm_if

# Configuration

*Note*

*Using DTC with UART is recommended to reduce the number of interrupts in the system.*

### Build Time Configurations for rm_cellular_comm_uart_aws

The following build time configurations are defined in fsp_cfg/middleware/rm_cellular_comm_uart_aws_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Selects if code for parameter checking is to be included in the build. |
| Receive Buffer | Size should be greater than zero | 256 | Temporary buffer for received bytes. When read function is called reamaining bytes will be read directly into buffer specified by read function argument. |
| Receive Transfer Size | Size should be greater than zero | 512 | The comm interface will break down UART reads into smaller read requests based on this number. For instance if this is set to 512 bytes then the interface will request 512 bytes at a time until the |

requested length is met or timeout. This is helpful for making use of DTC without having a long timeout period. AWS Cellular Interface Common->Comm Interface Receive Timeout is the timeout used for individual reads and should set accordingly based on this setting.

**Configurations for Networking > Cellular Comm Interface on UART (rm_cellular_comm_uart_aws)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_cellular_comm_uart0 | Module name. |

**Function Documentation**

◆ **RM_CELLULAR_COMM_UART_AWS_Open()**

CellularCommInterfaceError_t RM_CELLULAR_COMM_UART_AWS_Open ( rm_cellular_comm_uart_aws_instance_ctrl_t * *p_instance_ctrl*, CellularCommInterfaceReceiveCallback_t *receive_callback*, void * *pUserData* )

Implements open for CellularCommInterface_t

**Parameters**

| | | |
|---|---|---|
| [in] | receive_callback | Callback to call upon receiving data |
| [in] | pUserData | Pointer to user data to be passed through callback |
| [in] | p_instance_ctrl | Pointer to a rm_cellular_comm_uart_aws_instance_ctrl_t |

**Return values**

| | |
|---|---|
| IOT_COMM_INTERFACE_SUCCESS | Open succeeded |
| IOT_COMM_INTERFACE_BAD_PARAMETER | Bad parameter was passed in |
| IOT_COMM_INTERFACE_FAILURE | General failure |
| IOT_COMM_INTERFACE_DRIVER_ERROR | Lower level failure |

### ◆ RM_CELLULAR_COMM_UART_AWS_Send()

CellularCommInterfaceError_t RM_CELLULAR_COMM_UART_AWS_Send ( CellularCommInterfaceHandle_t *commInterfaceHandle*, const uint8_t * *pData*, uint32_t *dataLength*, uint32_t *timeoutMilliseconds*, uint32_t * *pDataSentLength* )

Implements send for CellularCommInterface_t. This function will block the calling thread until data is sent.

**Parameters**

| | | |
|---|---|---|
| [in] | commInterfaceHandle | CellularCommInterfaceHandle_t assigned in open |
| [in] | pData | Pointer to data to send |
| [in] | dataLength | Length of data to send |
| [in] | timeoutMilliseconds | Timeout in MS to wait for data to send |
| [in] | pDataSentLength | Actual length of data sent |

**Return values**

| | |
|---|---|
| IOT_COMM_INTERFACE_SUCCESS | Send succeeded |
| IOT_COMM_INTERFACE_BAD_PARAMETER | Bad parameter was passed in |
| IOT_COMM_INTERFACE_FAILURE | General failure |
| IOT_COMM_INTERFACE_DRIVER_ERROR | Lower level failure |
| IOT_COMM_INTERFACE_TIMEOUT | Send operation timed out |

### ◆ RM_CELLULAR_COMM_UART_AWS_Receive()

CellularCommInterfaceError_t RM_CELLULAR_COMM_UART_AWS_Receive ( CellularCommInterfaceHandle_t *commInterfaceHandle*, uint8_t * *pBuffer*, uint32_t *bufferLength*, uint32_t *timeoutMilliseconds*, uint32_t * *pDataReceivedLength* )

Implements receive for CellularCommInterface_t. This function will block the calling thread until data is received.

**Parameters**

| | | |
|---|---|---|
| [in] | commInterfaceHandle | CellularCommInterfaceHandle_t assigned in open |
| [in] | pBuffer | Pointer to buffer to receive to |
| [in] | bufferLength | Length of buffer |
| [in] | timeoutMilliseconds | Timeout in MS to wait for data reception |
| [in] | pDataReceivedLength | Actual length of data received |

**Return values**

| | |
|---|---|
| IOT_COMM_INTERFACE_SUCCESS | Receieve succeeded |
| IOT_COMM_INTERFACE_BAD_PARAMETER | Bad parameter was passed in |
| IOT_COMM_INTERFACE_FAILURE | General failure |
| IOT_COMM_INTERFACE_DRIVER_ERROR | Lower level failure |

### ◆ RM_CELLULAR_COMM_UART_AWS_Close()

CellularCommInterfaceError_t RM_CELLULAR_COMM_UART_AWS_Close ( CellularCommInterfaceHandle_t *commInterfaceHandle*)

Implements close for CellularCommInterface_t

**Parameters**

| | | |
|---|---|---|
| [in] | commInterfaceHandle | CellularCommInterfaceHandle_t assigned in open |

**Return values**

| | |
|---|---|
| IOT_COMM_INTERFACE_SUCCESS | Close succeeded |
| IOT_COMM_INTERFACE_BAD_PARAMETER | Bad parameter was passed in |
| IOT_COMM_INTERFACE_FAILURE | General failure |

### ◆ comm_uart_aws_callback()

| void comm_uart_aws_callback ( uart_callback_args_t * *p_args*) |
|---|
| Callback for UART driver<br><br>**Parameters**<br>[in] p_args — Arguments from UART RX/TX callback |

## 5.2.12.13 DA16XXX Transport Layer (rm_at_transport_da16xxx_uart)

Modules » Networking

### Functions

| | |
|---|---|
| void | rm_at_transport_da16xxx_uart_callback (uart_callback_args_t *p_args) |
| fsp_err_t | rm_at_transport_da16xxx_uartOpen (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_cfg_t const *const p_cfg) |
| fsp_err_t | rm_at_transport_da16xxx_uart_atCommandSendThreadSafe (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd) |
| fsp_err_t | rm_at_transport_da16xxx_uart_atCommandSend (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd) |
| fsp_err_t | rm_at_transport_da16xxx_uart_giveMutex (at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag) |
| fsp_err_t | rm_at_transport_da16xxx_uart_takeMutex (at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag) |
| size_t | rm_at_transport_da16xxx_uart_bufferRecv (at_transport_da16xxx_ctrl_t *const p_ctrl, const char *p_data, uint32_t length, uint32_t rx_timeout) |
| fsp_err_t | rm_at_transport_da16xxx_statusGet (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_status_t *p_status) |
| fsp_err_t | rm_at_transport_da16xxx_uartClose (at_transport_da16xxx_ctrl_t *const p_ctrl) |

### Detailed Description

Transport layer implementation for linking DA16XXX Drivers with Communications layer.

# Overview

This Transport Layer provides an abstraction interface between the DA16XXX driver and communications layers. Currently, the module supports only UART, but will support SPI in future.

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Features

- Configuring internal send/receive buffers.
- Providing thread-safe access to the communications interface (mutex) when sending AT commands.
- Implementing AT command send function based on user communication option (UART or SPI (future)). This function must be able to check the module response.
- Implement a receive function based on user communication option (UART or SPI (future)). Handles income data from the communications interface and stacks it on an internal buffer.

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

# Configuration

### Build Time Configurations for rm_at_transport_da16xxx_uart

The following build time configurations are defined in fsp_cfg/rm_at_transport_da16xxx_uart_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Size of RX buffer for CMD Port | Manual Entry | 3000 | |
| Semaphore maximum timeout | Manual Entry | 10000 | |
| Number of retries for AT commands | Manual Entry | 10 | |
| Module Reset Port | Refer to the RA Configuration tool for available options. | 06 | Specify the module reset pin port for the MCU. |
| Module Reset Pin | Refer to the RA Configuration tool for available options. | 03 | Specify the module reset pin for the MCU. |

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Interrupt Configuration

Refer to UART (r_sci_uart).

### Clock Configuration

Refer to UART (r_sci_uart).

### Pin Configuration

Refer to UART (r_sci_uart).

# Usage Notes

- This module is not designed to be used directly, users will access this module via the DA16XXX driver

### Limitations

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Data Structures

| | |
|---|---|
| struct | at_transport_da16xxx_extended_cfg_t |
| struct | at_transport_da16xxx_instance_ctrl_t |

### Data Structure Documentation

#### ◆ at_transport_da16xxx_extended_cfg_t

| struct at_transport_da16xxx_extended_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| const uint32_t | num_uarts | Number of UART interfaces to use. |
| const uart_instance_t * | uart_instances[AT_TRANSPORT_DA16XXX_CFG_MAX_NUMBER_UART_PORTS] | SCI UART instances. |
| const bsp_io_port_pin_t | reset_pin | Reset pin used for module. |

#### ◆ at_transport_da16xxx_instance_ctrl_t

| struct at_transport_da16xxx_instance_ctrl_t | |
|---|---|
| AT_TRANSPORT_DA16XXX private control block. DO NOT MODIFY. | |
| **Data Fields** | |
| at_transport_da16xxx_cfg_t const * | p_cfg |
| | Pointer to initial configurations. |

| | |
|---|---|
| | |
| uint32_t | num_uarts |
| | number of UARTS currently used for communication with module |
| | |
| uint32_t | curr_cmd_port |
| | Current UART instance index for AT commands. |
| | |
| uint32_t | open |
| | Flag to indicate if transport instance has been initialized. |
| | |
| uint8_t | cmd_rx_queue_buf [AT_TRANSPORT_DA16XXX_CFG_CMD_RX_BUF_SIZE] |
| | Command port receive buffer used by byte queue // FreeRTOS. |
| | |
| StreamBufferHandle_t | socket_byteq_hdl |
| | Socket stream buffer handle. |
| | |
| StaticStreamBuffer_t | socket_byteq_struct |
| | Structure to hold stream buffer info. |
| | |
| SemaphoreHandle_t | tx_sem |
| | Transmit binary semaphore handle. |
| | |
| SemaphoreHandle_t | rx_sem |
| | Receive binary semaphore handle. |
| | |
| SemaphoreHandle_t | uart_tei_sem [AT_TRANSPORT_DA16XXX_CFG_MAX_NUMBER_UART_PORTS] |
| | UART transmission end binary semaphore. |

| | |
|---|---|
| | |
| uart_instance_t * | uart_instance_objects [AT_TRANSPORT_DA16XXX_CFG_MAX_NUMBER_UART_PORTS] |
| | UART instance object. |
| | |
| const bsp_io_port_pin_t | reset_pin |
| | Reset pin used for module. |
| | |
| bool(* | p_callback )(at_transport_da16xxx_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |

## Function Documentation

### ◆ rm_at_transport_da16xxx_uart_callback()

void rm_at_transport_da16xxx_uart_callback ( uart_callback_args_t * *p_args*)

UART Callback routine.

**Parameters**

| | | |
|---|---|---|
| [in] | p_args | Pointer to uart callback structure. |

### ◆ rm_at_transport_da16xxx_uartOpen()

fsp_err_t rm_at_transport_da16xxx_uartOpen ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, at_transport_da16xxx_cfg_t const *const *p_cfg* )

Opens and configures the WIFI_DA16XXX Middleware module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_DA16XXX successfully configured. |
| FSP_ERR_ASSERTION | The parameter p_cfg or p_instance_ctrl is NULL. |
| FSP_ERR_OUT_OF_MEMORY | There is no more heap memory available. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_WIFI_INIT_FAILED | WiFi module initialization failed. |

### ◆ rm_at_transport_da16xxx_uart_atCommandSendThreadSafe()

fsp_err_t rm_at_transport_da16xxx_uart_atCommandSendThreadSafe ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, at_transport_da16xxx_data_t * *p_at_cmd* )

Send an AT command with testing for return response.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | p_at_cmd | Pointer to AT command data structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |

### ◆ rm_at_transport_da16xxx_uart_atCommandSend()

fsp_err_t rm_at_transport_da16xxx_uart_atCommandSend ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, at_transport_da16xxx_data_t * *p_at_cmd* )

Send and receive an AT command with testing for return. Thread-Safe

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | p_at_cmd | Pointer to Transport layer instance data structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |

### ◆ rm_at_transport_da16xxx_uart_giveMutex()

fsp_err_t rm_at_transport_da16xxx_uart_giveMutex ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, uint32_t *mutex_flag* )

Give the mutex for the send basic call.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | mutex_flag | Flags for the mutex. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred giving the mutex. |

### ◆ rm_at_transport_da16xxx_uart_takeMutex()

fsp_err_t rm_at_transport_da16xxx_uart_takeMutex ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, uint32_t *mutex_flag* )

Take the mutex for the send basic call.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | mutex_flag | Flags for the mutex. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred taking the mutex. |

### ◆ rm_at_transport_da16xxx_uart_bufferRecv()

size_t rm_at_transport_da16xxx_uart_bufferRecv ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, const char * *p_data*, uint32_t *length*, uint32_t *rx_timeout* )

Receive data from stream buffer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | p_data | Pointer to data. |
| [in] | length | Data length. |
| [in] | rx_timeout | Timeout for receiving data on the buffer. |

**Return values**

| | |
|---|---|
| Number | of bytes pulled from Streambuffer |
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_INVALID_DATA | Accuracy of data is not guaranteed |

### ◆ rm_at_transport_da16xxx_statusGet()

fsp_err_t rm_at_transport_da16xxx_statusGet ( at_transport_da16xxx_ctrl_t *const *p_ctrl*, at_transport_da16xxx_status_t * *p_status* )

Gets the status of the configured DA16xxx transport.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |

### ◆ rm_at_transport_da16xxx_uartClose()

fsp_err_t rm_at_transport_da16xxx_uartClose ( at_transport_da16xxx_ctrl_t *const *p_ctrl*)

Closes the AT Transport DA16XXX Middleware module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_DA16XXX successfully configured. |
| FSP_ERR_ASSERTION | The parameter p_cfg or p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The Transport layer instance is not open. |

## 5.2.12.14 EtherCAT SSC Port (rm_ethercat_ssc_port)

Modules » Networking

### Functions

fsp_err_t RM_ETHERCAT_SSC_PORT_Open (ethercat_ssc_port_ctrl_t *const p_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg)

EtherCAT Slave Controller is initialized with this function. This function includes PHY initialization and ESC EEPROM loading. Afterwards, EtherCAT communication begins. Also EtherCAT interrupts are permitted if the interrupts are used. In order to receive the EtherCAT, EtherCAT Sync0, EtherCAT Sync1 interrupt event, it's necessary to register a callback function. Implements ethercat_ssc_port_api_t::open. More...

| | |
|---|---|
| fsp_err_t | RM_ETHERCAT_SSC_PORT_Close (ethercat_ssc_port_ctrl_t *const p_ctrl) |

### Detailed Description

Middleware for EtherCAT Slave Stack Code (SSC) on RA microprocessors. This module implements the EtherCAT SSC Interface.

# Overview

This module provides the hardware port layer for EtherCAT Slave Stack Code. The control target is the EtherCAT Slave Controller (ESC). This module uses the EtherCAT slave stack code generated by the SSC tools provided by Beckhoff Automation GmbH. Users must register as a member of the EtherCAT Technology Group (ETG) to obtain SSC
tools.:
https://www.ethercat.org/en/downloads/downloads_01DCC32A10294F2EA866F7E46FB0285F.htm
After initializing this module, refer to the SSC API reference to use the EtherCAT slave stack. The hardware access layer API referenced from the stack are also implemented. Please include renesashw.h in the stack source file which has hardware access.

### Features

The EtherCAT SSC port module supports the following features:

- Number of EtherCAT ports: 2
- Interrupts: EtherCAT interrupt, Sync0 interrupt, Sync1 interrupt
- Hardware access layer specified by SSC (including timer)
- Initialize ESC and PHY to enable EtherCAT communication

# Configuration

### Build Time Configurations for rm_ethercat_ssc_port

The following build time configurations are defined in fsp_cfg/rm_ethercat_ssc_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Phy Link Polarity0 | • Active High<br>• Active Low | Active Low | Set the signal porality of CAT0_LINKSTA pin. |
| Phy Link Polarity1 | • Active High<br>• Active Low | Active High | Set the signal porality of CAT1_LINKSTA pin. |

### Configurations for Networking > EtherCAT SSC Port (rm_ethercat_ssc_port)

This module can be added to the Stacks tab via New Stack > Networking > EtherCAT SSC Port (rm_ethercat_ssc_port).

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| PHY | | | |
| Reset Signal Hold Time (ms) | Specify a value between 0x1 and 0xFFFFFFFF. | 1 | Time to keep the PHY reset signal low. |
| Reset Wait Time (us) | Specify a value between 0x1 and 0xFFFFFFFF. | 15000 | Wait time to access the PHY register after the PHY reset signal is released. |
| Offset Address | Specify a value between 0x0 and 0xFF. | 0 | The offset address of PHY in case of using the EtherCAT. |
| Delay Time of TX Signal | | | |
| Port 0 | • 0 ns<br>• 10 ns<br>• 20 ns<br>• 30 ns | 0 ns | Set the amount of delay time for TX signal of the Port 0 |
| Port 1 | • 0 ns<br>• 10 ns<br>• 20 ns<br>• 30 ns | 0 ns | Set the amount of delay time for TX signal of the Port 1 |
| Interrupts | | | |
| EtherCAT Interrupt Priority | MCU Specific Options | | Select the EtherCAT interrupt priority. |
| EtherCAT SYNC0 Interrupt Priority | MCU Specific Options | | Select the EtherCAT SYNC0 interrupt priority. |
| EtherCAT SYNC1 Interrupt Priority | MCU Specific Options | | Select the EtherCAT SYNC1 interrupt priority. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |
| Name | Name must be a valid C symbol | g_ethercat_ssc_port0 | Module name. |
| EEPROM Size | • Under 32Kbits<br>• Over 32Kbits | Under 32Kbits | Specify whether to use Normal or Inverted. |

### Clock Configuration

The ESC uses the ESCCLK which is generated by the clock generation circuit and the ESCCLK divided by 4 as CLK100 and CLK25 of the ESC IP Core, respectively.

Set the registers in the clock generation circuit so that the frequency of the ESCCLK becomes 100 MHz. Also, set the registers that the frequency of the ETHPHYCLK signal which is used as the operating clock of a PHY-LSI becomes 25 MHz.

### Pin Configuration

The RM_ETHERCAT_SSC_PORT module is used for configuring pins.

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the RM_ETHERCAT_SSC_PORT:

- Port configuration is EtherCAT port only.

### ETHERCAT SSC Port Basic Example

This is a basic example of minimal use of the ETHERCAT SSC Port in an application.

```
void ethercat_ssc_port_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module before using the EtherCAT Slave Stack. */
 /* The instance of EtherCAT SSC Port instance can be accessed as
gp_ether_cat_ssc_port */
    err = RM_ETHERCAT_SSC_PORT_Open(gp_ethercat_ssc_port->p_ctrl,
gp_ethercat_ssc_port->p_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 /* Close the module after using the EtherCAT Slave Stack. */
 RM_ETHERCAT_SSC_PORT_Close(gp_ethercat_ssc_port->p_ctrl);
}
/* Example callback called when EtherCAT interrupt. */
void ethercat_ssc_port_callback (ethercat_ssc_port_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case ETHERCAT_SSC_PORT_EVENT_ESC_CAT_INTERRUPT:
      {
/* ESC CAT Interrupt application */
 break;
       }
 case ETHERCAT_SSC_PORT_EVENT_ESC_SYNC0_INTERRUPT:
      {
```

```
/* ESC SYNC0 Interrupt application */
 break;
       }
 case ETHERCAT_SSC_PORT_EVENT_ESC_SYNC1_INTERRUPT:
      {
/* ESC SYNC1 Interrupt application */
 break;
       }
 default:
      {
 /* Do Nothing! */
       }
    }
}
```

### Data Structures

| | |
|---|---|
| struct | ethercat_ssc_port_instance_ctrl_t |
| struct | ethercat_ssc_port_extend_cfg_t |

### Enumerations

| | |
|---|---|
| enum | ethercat_ssc_port_eeprom_size_t |
| enum | ethercat_ssc_port_tx_signal_delay_t |

## Data Structure Documentation

### ◆ ethercat_ssc_port_instance_ctrl_t

| struct ethercat_ssc_port_instance_ctrl_t | | |
|---|---|---|
| EtherCAT SSC port control block. DO NOT INITIALIZE. Initialization occurs when ethercat_ssc_port_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Flag to indicate if EtherCAT SSC port instance has been initialized. |
| uint16_t | sys_time | System time of EtherCAT Slave Controller. |
| ethercat_ssc_port_cfg_t const * | p_cfg | Pointer to initial configurations. |
| R_ESC_INI_Type * | p_reg_esc_ini | Pointer to EtherCAT Initial |

| | | |
|---|---|---|
| | | Configuration registers. |
| R_ESC_Type * | p_reg_esc | Pointer to EtherCAT Slave Controller registers. |

### ◆ ethercat_ssc_port_extend_cfg_t

| struct ethercat_ssc_port_extend_cfg_t | | |
|---|---|---|
| Extended configuration | | |
| Data Fields | | |
| ethercat_ssc_port_eeprom_size_t | eeprom_size | EEPROM memory size. |
| ethercat_ssc_port_tx_signal_delay_t | tx_signal_delay_port0 | Port 0 TX signal delay time. |
| ethercat_ssc_port_tx_signal_delay_t | tx_signal_delay_port1 | Port 1 TX signal delay time. |
| ether_phy_instance_t const * | p_ether_phy_instance[BSP_FEATURE_ESC_NUM_PORTS] | Pointer to ETHER_PHY instance. |

## Enumeration Type Documentation

### ◆ ethercat_ssc_port_eeprom_size_t

| enum ethercat_ssc_port_eeprom_size_t | |
|---|---|
| EEPROM memory size of the EtherCAT | |
| Enumerator | |
| ETHERCAT_SSC_PORT_EEPROM_SIZE_UNDER_32KBIT | Up to 16KBit. |
| ETHERCAT_SSC_PORT_EEPROM_SIZE_OVER_32KBIT | 32KBit to 4MBit |

### ◆ ethercat_ssc_port_tx_signal_delay_t

| enum ethercat_ssc_port_tx_signal_delay_t | |
|---|---|
| Delay time of TX signal | |
| Enumerator | |
| ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_00NS | 0 ns |
| ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_10NS | 10 ns |
| ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_20NS | 20 ns |
| ETHERCAT_SSC_PORT_TX_SIGNAL_DELAY_30NS | 30 ns |

### Function Documentation

#### ◆ RM_ETHERCAT_SSC_PORT_Open()

fsp_err_t RM_ETHERCAT_SSC_PORT_Open ( ethercat_ssc_port_ctrl_t *const *p_ctrl*, ethercat_ssc_port_cfg_t const *const *p_cfg* )

EtherCAT Slave Controller is initialized with this function. This function includes PHY initialization and ESC EEPROM loading. Afterwards, EtherCAT communication begins. Also EtherCAT interrupts are permitted if the interrupts are used. In order to receive the EtherCAT, EtherCAT Sync0, EtherCAT Sync1 interrupt event, it's necessary to register a callback function. Implements ethercat_ssc_port_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Opened successfully |
| FSP_ERR_ASSERTION | A required input pointer is NULL |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened |
| FSP_ERR_INVALID_ARGUMENT | Irq number lower then 0. |
| FSP_ERR_INVALID_HW_CONDITION | ESC EEPROM loading error |
| FSP_ERR_TIMEOUT | ESC EEPROM loading timeout error |

#### ◆ RM_ETHERCAT_SSC_PORT_Close()

fsp_err_t RM_ETHERCAT_SSC_PORT_Close ( ethercat_ssc_port_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ethercat_ssc_port_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to the control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

## 5.2.12.15 Ethercat (r_ethercat_phy)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_ETHERCAT_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br><br>Resets Ethercat PHY device. Implements ether_phy_api_t::open. |

| | |
|---|---|
| | More... |
| fsp_err_t | R_ETHERCAT_PHY_Close (ether_phy_ctrl_t *const p_ctrl)<br>Close Ethercat PHY device. Implements ether_phy_api_t::close. More... |
| fsp_err_t | R_ETHERCAT_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl)<br>Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More... |
| fsp_err_t | R_ETHERCAT_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)<br>Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More... |
| fsp_err_t | R_ETHERCAT_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl)<br>Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More... |
| fsp_err_t | R_ETHERCAT_PHY_ChipInit (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br>Initialize Ethercat PHY device. Implements ether_phy_api_t::chipInit. More... |
| fsp_err_t | R_ETHERCAT_PHY_Read (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data)<br>Read data from register of PHY-LSI . Implements ether_phy_api_t::read. More... |
| fsp_err_t | R_ETHERCAT_PHY_Write (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)<br>Write data to register of PHY-LSI . Implements ether_phy_api_t::write. More... |

## Detailed Description

The Ethercat PHY module (r_ethercat_phy) provides an API for standard Ethercat PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY

Interface.

# Overview

The Ethercat PHY module is used to setup and manage an external Ethercat PHY device for use with the on-chip Ethercat Slave Controller (ESC) peripheral. It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

### Features

The Ethercat PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

# Configuration

### Build Time Configurations for r_ethercat_phy

The following build time configurations are defined in fsp_cfg/r_ethercat_phy_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| ICS1894 Target | • Disabled<br>• Enabled | Disabled | Select whether to use ICS1894 Phy LSI or not. |
| User Own Target | • Disabled<br>• Enabled | Disabled | Select whether to use User own Phy LSI or not. |

### Configurations for Networking > Ethernet (r_ethercat_phy)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ethercat_phy). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ethercat_phy0 | Module name. |
| Channel | • 0<br>• 1 | 0 | Select the Ethernet controller channel number. |
| PHY-LSI Address | Specify a value between 0 and 31. | 0 | Specify the address of the PHY-LSI used. |
| PHY-LSI Reset Completion Timeout | Specify a value between 0x1 and | 0x00020000 | Specify the number of times to read the PHY- |

| | | | |
|---|---|---|---|
| | 0xFFFFFFFF. | | LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used. |
| Select MII type | MII | module.driver.ether_phy.mii_type.rmii | Specify to use MII. |
| PHY-LSI Type | • Kit Component<br>• DEFAULT<br>• ICS1894<br>• User own PHY | Kit Component | Select the Phy LSI target. Selecting 'Kit Component' will automatically choose the correct option when using a Renesas development board. |
| Target Init Function | Name must be a valid C symbol | NULL | Set the initial function of the PHY-LSI, When using your own PHY-LSI. |
| Target Link Partner Ability Get Function | Name must be a valid C symbol | NULL | Set the link partner ability get function of the PHY-LSI, When using your own PHY-LSI. |
| MII Register Access Wait-time | Specify a value between 0x1 and 0x7FFFFFFF. | 8 | Specify the bit timing for MII register accesses during PHY initialization. This value should be adjusted experimentally based on the PHY-LSI used. |
| Flow Control | • Disable<br>• Enable | Disable | Select whether to enable or disable flow control. |

# Usage Notes

*Note*

*See the example below for details on how to initialize the Ethercat PHY module.*

### Accessing the MII and RMII Registers

Use the PIR register to access the MII and RMII registers in the PHY-LSI. Serial data in the MII and RMII management frame format is transmitted and received through the ET0_MDC and ET0_MDIO pins controlled by software.

### MII and RMII management frame format

The below table lists the MII and RMII management frame formats.

| Access type | MII and RMII management frame | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Item | PRE | ST | OP | PHYAD | REGAD | TA | DATA | IDLE |
| | Number of bits | 32 | 2 | 2 | 5 | 5 | 2 | 16 | 1 |
| Read | | 1...1 | 01 | 10 | 00001 | RRRRR | Z0 | DDDDDDDDDDDDDDDD | Z |
| Write | | 1...1 | 01 | 01 | 00001 | RRRRR | 10 | DDDDDDDDDDDDDDDD | Z |

*Note*

*- PRE (preamble): Send 32 consecutive 1s.*
*- ST (start of frame): Send 01b.*
*- OP (operation code): Send 10b for read or 01b for write.*
*- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.*
*- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.*
*- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.*
*Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).*
*(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)*
*- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.*
*- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write*
*operation (Z is output). No control is required, because a bus was already released during a read operation.*

### Limitations

- The r_ethercat_phy module may need to be customized for PHY devices other than the ones currently supported. Use the existing code as a starting point for creating a custom implementation. Supported PHY devices are as follows.
  - ICS1894

# Examples

### ETHERCAT PHY Basic Example

This is a basic example of minimal use of the ETHERCAT PHY in an application.

```
void ethercat_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ethercat_phy0_ctrl.open   = 0U;
    g_ethercat_phy0_cfg.channel = 0;
```

```
/* Initializes the module. */
   err = R_ETHERCAT_PHY_Open(&g_ethercat_phy0_ctrl, &g_ethercat_phy0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Start auto negotiation. */
   err = R_ETHERCAT_PHY_StartAutoNegotiate(&g_ethercat_phy0_ctrl);
   assert(FSP_SUCCESS == err);
/* Polling until link is established. */
while (FSP_SUCCESS != R_ETHERCAT_PHY_LinkStatusGet(&g_ethercat_phy0_ctrl))
   {
/* Do nothing */
   }
/* Get link partner ability from phy interface. */
   err = R_ETHERCAT_PHY_LinkPartnerAbilityGet(&g_ethercat_phy0_ctrl,
                                              &g_ethercat_phy0_line_speed_duplex,
                                              &g_ethercat_phy0_local_pause,
                                              &g_ethercat_phy0_partner_pause);
   assert(FSP_SUCCESS == err);
/* Check current link status. */
   err = R_ETHERCAT_PHY_LinkStatusGet(&g_ethercat_phy0_ctrl);
   assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | ethercat_phy_instance_ctrl_t |
| struct | ethercat_phy_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | ethercat_phy_interface_status_t |

## Data Structure Documentation

### ◆ ethercat_phy_instance_ctrl_t

| struct ethercat_phy_instance_ctrl_t |
|---|
| ETHER PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called. |

| Data Fields | | |
|---|---|---|
| uint32_t | open | Used to determine if the channel is configured. |
| ether_phy_cfg_t const * | p_ether_phy_cfg | Pointer to initial configurations. |
| uint32_t | local_advertise | Capabilities bitmap for local advertising. |
| ethercat_phy_interface_status_t | interface_status | Initialized status of ETHER PHY interface. |

### ◆ ethercat_phy_extended_cfg_t

| struct ethercat_phy_extended_cfg_t | |
|---|---|
| ETHERCAT PHY extended configuration. | |
| **Data Fields** | |
| void(* | p_target_init )(ethercat_phy_instance_ctrl_t *p_instance_ctrl) |
| | Pointer to callback that is called to initialize the target. |
| | |
| bool(* | p_target_link_partner_ability_get )(ethercat_phy_instance_ctrl_t *p_instance_ctrl, uint32_t line_speed_duplex) |
| | Pointer to callback that is called to get the link partner ability. |
| | |

## Enumeration Type Documentation

### ◆ ethercat_phy_interface_status_t

| enum ethercat_phy_interface_status_t | |
|---|---|
| Initialization state for read/write | |
| Enumerator | |
| ETHERCAT_PHY_INTERFACE_STATUS_UNINITIALIZED | ETHER PHY interface is uninitialized. |
| ETHERCAT_PHY_INTERFACE_STATUS_INITIALIZED | ETHER PHY interface is initialized. |

## Function Documentation

### ◆ R_ETHERCAT_PHY_Open()

fsp_err_t R_ETHERCAT_PHY_Open ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Resets Ethercat PHY device. Implements ether_phy_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |

### ◆ R_ETHERCAT_PHY_Close()

fsp_err_t R_ETHERCAT_PHY_Close ( ether_phy_ctrl_t *const *p_ctrl*)

Close Ethercat PHY device. Implements ether_phy_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHERCAT_PHY_StartAutoNegotiate()

fsp_err_t R_ETHERCAT_PHY_StartAutoNegotiate ( ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHERCAT_PHY successfully starts auto-negotiate. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_INVALID_ARGUMENT | Register address is incorrect |
| FSP_ERR_INVALID_MODE | Command is invalid. |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHERCAT_PHY_LinkPartnerAbilityGet()

fsp_err_t R_ETHERCAT_PHY_LinkPartnerAbilityGet ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause* )

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHERCAT_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | The auto-negotiation isn't completed |
| FSP_ERR_INVALID_ARGUMENT | Status register address is incorrect |
| FSP_ERR_INVALID_MODE | Command is invalid. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHERCAT_PHY_LinkStatusGet()

fsp_err_t R_ETHERCAT_PHY_LinkStatusGet ( ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHERCAT_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_INVALID_ARGUMENT | Status register address is incorrect |
| FSP_ERR_INVALID_MODE | Command is invalid. |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHERCAT_PHY_ChipInit()

fsp_err_t R_ETHERCAT_PHY_ChipInit ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Initialize Ethercat PHY device. Implements ether_phy_api_t::chipInit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PHY device initialized successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block or configuration structure is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_TIMEOUT | PHY-LSI Reset wait timeout. |

### ◆ R_ETHERCAT_PHY_Read()

fsp_err_t R_ETHERCAT_PHY_Read ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *const *p_data* )

Read data from register of PHY-LSI . Implements ether_phy_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHERCAT_PHY successfully read data. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address is not a valid size |
| FSP_ERR_INVALID_MODE | Command is invalid. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHERCAT_PHY_Write()

fsp_err_t R_ETHERCAT_PHY_Write ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *data* )

Write data to register of PHY-LSI . Implements ether_phy_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHERCAT_PHY successfully write data. |
| FSP_ERR_ASSERTION | Pointer to ETHERCAT_PHY control block is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address or data is not a valid size |
| FSP_ERR_INVALID_MODE | Command is invalid. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

## 5.2.12.16 Ethernet (r_ether)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_ETHER_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)<br><br>After ETHERC, EDMAC and PHY-LSI are reset in software, an auto |

| | |
|---|---|
| | negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open. More... |
| fsp_err_t | R_ETHER_Close (ether_ctrl_t *const p_ctrl)<br>Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More... |
| fsp_err_t | R_ETHER_BufferRelease (ether_ctrl_t *const p_ctrl)<br>Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease. More... |
| fsp_err_t | R_ETHER_RxBufferUpdate (ether_ctrl_t *const p_ctrl, void *const p_buffer)<br>Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate. More... |
| fsp_err_t | R_ETHER_LinkProcess (ether_ctrl_t *const p_ctrl)<br>The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess. More... |
| fsp_err_t | R_ETHER_WakeOnLANEnable (ether_ctrl_t *const p_ctrl)<br>The setting of ETHERC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable. More... |
| fsp_err_t | R_ETHER_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes)<br>Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read. More... |
| fsp_err_t | R_ETHER_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)<br>Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write. More... |

| | |
|---|---|
| fsp_err_t | R_ETHER_TxStatusGet (ether_ctrl_t *const p_ctrl, void *const p_buffer_address) |
| fsp_err_t | R_ETHER_CallbackSet (ether_ctrl_t *const p_api_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory) |

### Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface.

# Overview

This module performs Ethernet frame transmission and reception using an Ethernet controller and an Ethernet DMA controller.

### Features

The Ethernet module supports the following features:

- Transmit/receive processing
- Optional zero-copy buffering
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support

# Configuration

### Build Time Configurations for r_ether

The following build time configurations are defined in fsp_cfg/r_ether_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| ET0_LINKSTA Pin Status Flag | • Fall -> Rise<br>• Rise -> Fall | Fall -> Rise | Specify the polarity of the link signal output by the PHY-LSI. When 0 is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When 1 is specified, link-up and link-down correspond |

| | | | |
|---|---|---|---|
| | | | respectively to the rise and fall of the LINKSTA signal. |
| Link Signal Change Flag | • Unused<br>• Used | Unused | Use LINKSTA signal for detect link status changes 0 = unused (use PHY-LSI status register) 1 = use (use LINKSTA signal) |

**Configurations for Networking > Ethernet (r_ether)**

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ether). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_ether0 | Module name. |
| Channel | 0 | 0 | Select the ether channel number. |
| MAC address | Must be a valid MAC address | 00:11:22:33:44:55 | MAC address of this channel. |
| Zero-copy Mode | • Disable<br>• Enable | Disable | Enable or disable zero-copy mode. |
| Flow control functionality | • Disable<br>• Enable | Disable | Enable or disable flow control. |
| Filters | | | |
| Multicast Mode | • Disable<br>• Enable | Enable | Enable or disable multicast frame reception. |
| Promiscuous Mode | • Disable<br>• Enable | Disable | Enable this option to receive packets addressed to other NICs. |
| Broadcast filter | Must be a valid non-negative integer with maximum configurable value of 65535. | 0 | Limit of the number of broadcast frames received continuously |
| Buffers | | | |
| Number of TX buffer | Must be an integer from 1 to 8 | 1 | Number of transmit buffers |
| Number of RX buffer | Must be an integer from 1 to 8 | 1 | Number of receive buffers |

| | | | |
|---|---|---|---|
| Allocate RX buffer | • Disable<br>• Enable | Enable | Allocates the RX buffer when generating the configuration structure |
| Buffer size | Must be at least 1514 which is the maximum Ethernet frame size. | 1514 | Size of Ethernet buffer |
| Padding size | • Disable<br>• 1 Byte<br>• 2 Bytes<br>• 3 Bytes | Disable | The padding size that is automatically inserted into the received packets |
| Padding offset | Must be less than 64 bytes. | 0 | The offset into a receive buffer to insert padding bytes. |
| Interrupts | | | |
| EESR event | Refer to the RA Configuration tool for available options. | • RFOF<br>• RDE<br>• FR<br>• TFUF<br>• TDE<br>• TC | Select event list for each bit of EESR. |
| ECSR event | • ICD<br>• MPD<br>• LCHNG<br>• PSRTO<br>• BFR | | Select event list for each bit of ECSR. |
| Interrupt priority | MCU Specific Options | | Select the EDMAC interrupt priority. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |

### Interrupt Configuration

The first R_ETHER_Open function call sets EINT interrupts. The user could provide callback function which would be invoked when EINT interrupt handler has been completed. The callback arguments will contain information about a channel number, the ETHERC and EDMAC status, the event code, and a pointer to the user defined context.

### Callback Configuration

The user could provide callback function which would be invoked when either a magic packet or a link signal change is detected. When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.4 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value should be copied into, for example, a global variable.

### Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

| MCU Group | Peripheral Clock |
|---|---|
| RA4E2 | PCLKA |
| RA4T1 | PCLKA |
| RA6E1 | PCLKA |
| RA6E2 | PCLKA |
| RA6M2 | PCLKA |
| RA6M3 | PCLKA |
| RA6M4 | PCLKA |
| RA6M5 | PCLKA |
| RA6T3 | PCLKA |

*Note*

1. *When using ETHERC, the PCLKA frequency is in the range 12.5 MHz <= PCLKA <= 120 MHz.*
2. *When using ETHERC, PCLKA = ICLK.*

### Pin Configuration

To use the Ethernet module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). Please perform the pin setting before calling the R_ETHER_Open function.

# Usage Notes

### Ethernet Frame Format

The Ethernet module supports the Ethernet II/IEEE 802.3 frame format.

### Frame Format for Data Transmission and Reception

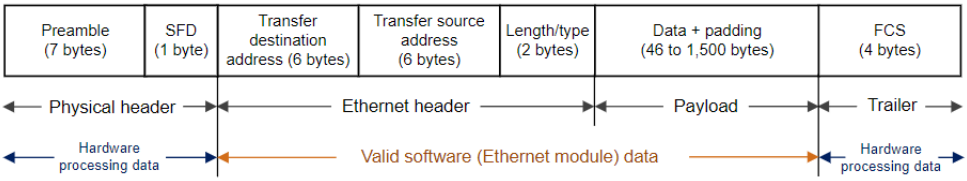



Figure 315: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

### PAUSE Frame Format

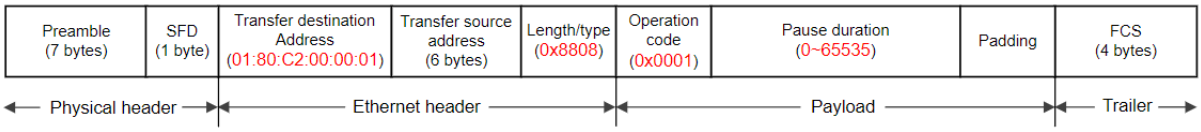



Figure 316: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

### Magic Packet Frame Format

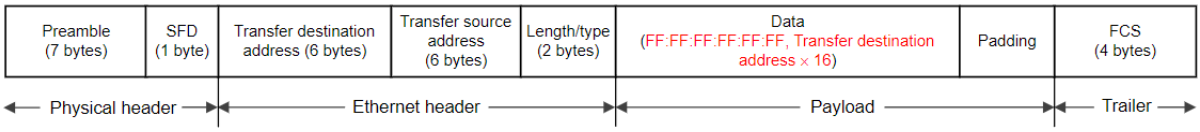



Figure 317: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

### Limitations

### Memory alignment limitation for Ethernet buffer

The Ethernet Driver has several alignment constraints:

- 16-byte alignment for the descriptor
- 32-byte aligned read buffer for R_ETHER_RxBufferUpdate when zero copy mode is enabled

### Functional limitations in TrustZone Security Extensions

The Ethernet Driver has several security constraints:

| MCU | Has Security Extension | Support Flat project | Support TrustZone project | |
|---|---|---|---|---|
| | | | Secure | Non-Secure |
| RA6M2 | - | x | - | - |
| RA6M3 | - | x | - | - |
| RA6M4 | - *1 | x | - | x |
| RA6M5 | - *1 | x | - | x |
| RA8M1 | x | x | x | x |

*Note*

*1. ETHERC/EDMAC is always Non-secure peripheral in this MCU.*

# Examples

## ETHER Basic Example

This is a basic example of minimal use of the ETHER in an application.

*Note*

*In this example zero-copy mode is disabled and there are no restrictions on buffer alignment.*

```
#define ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE (1514)
#define ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define ETHER_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x02
#define ETHER_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define ETHER_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

/* Receive data buffer */
static uint8_t gp_read_buffer[ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE] = {0};
/* Transmit data buffer */
static uint8_t gp_send_data[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
    uint32_t read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
```

```
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    assert(FSP_SUCCESS == err);
 /* received data copy to user buffer from internal buffer. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
}
```

### ETHER Advanced Example

The example demonstrates using send and receive function in zero copy mode. Transmit buffers must be 32-byte aligned and the receive buffer must be released once its contents have been used.

```
#define ETHER_EXAMPLE_FLAG_ON (1U)
#define ETHER_EXAMPLE_FLAG_OFF (0U)
#define ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK (1UL << 18)
#define ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK (1UL << 21)
#define ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK (1UL << 1)
#define ETHER_EXAMPLE_ALIGNMENT_32_BYTE (32)
static volatile uint32_t g_example_receive_complete = 0;
static volatile uint32_t g_example_transfer_complete = 0;
```

```
static uint8_t gp_send_data_internal[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_example_callback (ether_callback_args_t * p_args) {
 switch (p_args->event)
    {
 case ETHER_EVENT_TX_COMPLETE:
     {
           g_example_transfer_complete = ETHER_EXAMPLE_FLAG_ON;
 break;
      }
 case ETHER_EVENT_RX_COMPLETE:
     {
           g_example_receive_complete = ETHER_EXAMPLE_FLAG_ON;
 break;
      }
 default:
     {
      }
    }
}
void ether_advanced_use_internal_buffer_example (void) {
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
 static uint8_t * p_read_buffer_nocopy;
    uint32_t         read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
    g_ether0_cfg.zerocopy   = ETHER_ZEROCOPY_ENABLE;
    g_ether0_cfg.p_callback = (void (*)(ether_callback_args_t
```

```
*))ether_example_callback;
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set user buffer to TX descriptor and enable transmission. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data_internal, sizeof
(gp_send_data_internal));
 if (FSP_SUCCESS == err)
    {
 /* Wait for the transmission to complete. */
 /* Data array should not change in zero copy mode until transfer complete. */
 while (ETHER_EXAMPLE_FLAG_ON != g_example_transfer_complete)
     {
           ;
      }
    }
 /* Get receive buffer from RX descriptor. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) &p_read_buffer_nocopy,
&read_data_size);
    assert(FSP_SUCCESS == err);
 /* Process received data here */
 /* Release receive buffer to RX descriptor. */
    err = R_ETHER_BufferRelease(&g_ether0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
```

```
}
```

```
#define ETHER_EXAMPLE_ALIGNMENT_32_BYTE (32)
#define ETHER_EXAMPLE_ETHERNET_FRAME_PAYLOAD_OFFSET (14)
/* The data buffer must be 32-byte aligned when using zero copy mode. */
static uint8_t gp_send_data_external[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
typedef struct st_buffer_node
{
    uint8_t               * p_buffer;
 struct st_buffer_node * p_next;
} buffer_node_t;
void ether_advanced_use_external_buffer_example (void) {
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
    uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
    uint8_t       * p_tx_buffer          = NULL;
    uint8_t       * p_rx_buffer          = NULL;
    uint8_t       * p_rx_allocate_buffer = NULL;
    uint8_t       * p_tx_last_sent_buffer = NULL;
    buffer_node_t * p_tx_buffer_head;
    buffer_node_t * p_tx_buffer_tail;
    uint32_t read_data_size = 0;
    uint8_t i;
    g_ether0_cfg.p_mac_address = mac_address_source;
    g_ether0_cfg.zerocopy         = ETHER_ZEROCOPY_ENABLE;
    g_ether0_cfg.pp_ether_buffers = NULL;
 /* Create ring buffer structure to manage transmit buffer.*/
    p_tx_buffer_head = (buffer_node_t *) malloc(sizeof(buffer_node_t));
```

```
   p_tx_buffer_tail = p_tx_buffer_head;
for (i = 0; i < g_ether0_cfg.num_tx_descriptors - 1; i++)
   {
      p_tx_buffer_tail->p_buffer = NULL;
      p_tx_buffer_tail->p_next   = (buffer_node_t *) malloc(sizeof(buffer_node_t));
      p_tx_buffer_tail          = p_tx_buffer_tail->p_next;
   }
   p_tx_buffer_tail->p_buffer = NULL;
   p_tx_buffer_tail->p_next   = p_tx_buffer_head;
/* Open the ether instance with initial configuration. */
   err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
   i = 0;
/* Initialize receive buffer in Ethernet driver. */
while (i < g_ether0_cfg.num_rx_descriptors)
   {
if (posix_memalign((void **) &p_rx_allocate_buffer, ETHER_EXAMPLE_ALIGNMENT_32_BYTE,
                           g_ether0_cfg.ether_buffer_size * sizeof(char)))
     {
/* Set receive buffer to Ethernet driver. */
           err = R_ETHER_RxBufferUpdate(&g_ether0_ctrl, (void *)
p_rx_allocate_buffer);
if (FSP_SUCCESS == err)
     {
               i++;
        }
      }
else
     {
           assert(0);
      }
   }
do
```

```
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 while (1)
    {
 if (NULL == p_tx_buffer_tail->p_buffer)
     {
 /* Allocate memory to transmit buffer */
           p_tx_buffer = (uint8_t *) malloc(sizeof(gp_send_data_external));
 /* Process transmit data here. */
           memcpy(p_tx_buffer, gp_send_data_external, sizeof
(gp_send_data_external));
           gp_send_data_external[ETHER_EXAMPLE_ETHERNET_FRAME_PAYLOAD_OFFSET]++;
 /* Set user buffer to TX descriptor and enable transmission. */
           err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data_external,
sizeof(gp_send_data_external));
 /* Register transmit buffer to ring buffer. */
 if (FSP_SUCCESS == err)
     {
               p_tx_buffer_tail->p_buffer = p_tx_buffer;
        p_tx_buffer_tail          = p_tx_buffer_tail->p_next;
        }
 else
     {
 /* Release transmit buffer. */
               free(p_tx_buffer);
        }
      }
 /* Get receive buffer from RX descriptor. */
      err = R_ETHER_Read(&g_ether0_ctrl, (void *) &p_rx_buffer, &read_data_size);
 if (FSP_SUCCESS == err)
```

```
     {
 /* Allocate new receive buffer and update receive buffer to RX descriptor. */
 if (0 ==
                posix_memalign((void **) &p_rx_allocate_buffer,
ETHER_EXAMPLE_ALIGNMENT_32_BYTE,
                            g_ether0_cfg.ether_buffer_size * sizeof(char)))
     {
 R_ETHER_RxBufferUpdate(&g_ether0_ctrl, p_rx_allocate_buffer);
        }
 else
     {
             assert(0);
        }
 /* Process received data here. */
 /* Release receive buffer. */
           free(p_rx_buffer);
      }
 /* Release all transmitted buffer from the ring buffer. */
 if (FSP_SUCCESS == R_ETHER_TxStatusGet(&g_ether0_ctrl, (void *)
&p_tx_last_sent_buffer))
     {
           buffer_node_t * p_tx_buffer_current = p_tx_buffer_head;
 for (i = 0; i < g_ether0_cfg.num_tx_descriptors; i++)
     {
 if (p_tx_last_sent_buffer == p_tx_buffer_current->p_buffer)
     {
 do
     {
                     free(p_tx_buffer_head->p_buffer);
                     p_tx_buffer_head->p_buffer = NULL;
                     p_tx_buffer_head          = p_tx_buffer_head->p_next;
        } while (p_tx_buffer_head != p_tx_buffer_current);
                free(p_tx_buffer_head->p_buffer);
                p_tx_buffer_head->p_buffer = NULL;
```

```
break;
        }
                p_tx_buffer_current = p_tx_buffer_current->p_next;
        }
      }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | ether_instance_descriptor_t |
| struct | ether_extended_cfg_t |
| struct | ether_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ether_previous_link_status_t |
| enum | ether_link_change_t |
| enum | ether_magic_packet_t |
| enum | ether_link_establish_status_t |
| enum | ether_eesr_event_mask_t |
| enum | ether_ecsr_event_mask_t |

## Data Structure Documentation

### ◆ ether_instance_descriptor_t

| struct ether_instance_descriptor_t |
|---|
| EDMAC descriptor as defined in the hardware manual. Structure must be packed at 1 byte. |

### ◆ ether_extended_cfg_t

| struct ether_extended_cfg_t | | |
|---|---|---|
| ETHER extension configures the buffer descriptor for ETHER. | | |
| Data Fields | | |
| ether_instance_descriptor_t * | p_rx_descriptors | Receive descriptor buffer pool. |
| ether_instance_descriptor_t * | p_tx_descriptors | Transmit descriptor buffer pool. |

| | | |
|---|---|---|
| uint32_t | eesr_event_filter | Filter for EESR related event. |
| uint8_t | ecsr_event_filter | Filter for ECSR related event. |

### ◆ ether_instance_ctrl_t

| struct ether_instance_ctrl_t | |
|---|---|
| ETHER control block. DO NOT INITIALIZE. Initialization occurs when ether_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| ether_cfg_t const * | p_ether_cfg |
| | Pointer to initial configurations. |
| | |
| ether_instance_descriptor_t * | p_rx_descriptor |
| | Pointer to the currently referenced transmit descriptor. |
| | |
| ether_instance_descriptor_t * | p_tx_descriptor |
| | Pointer to the currently referenced receive descriptor. |
| | |
| void * | p_reg_etherc |
| | Base register of ethernet controller for this channel. |
| | |
| void * | p_reg_edmac |
| | Base register of EDMA controller for this channel. |
| | |
| ether_previous_link_status_t | previous_link_status |
| | Previous link status. |
| | |

| | |
|---|---|
| ether_link_change_t | link_change |
| | status of link change |
| | |
| ether_magic_packet_t | magic_packet |
| | status of magic packet detection |
| | |
| ether_link_establish_status_t | link_establish_status |
| | Current Link status. |
| | |

### Enumeration Type Documentation

#### ◆ ether_previous_link_status_t

enum ether_previous_link_status_t

| Enumerator | |
|---|---|
| ETHER_PREVIOUS_LINK_STATUS_DOWN | Previous link status is down. |
| ETHER_PREVIOUS_LINK_STATUS_UP | Previous link status is up. |
| ETHER_PREVIOUS_LINK_STATUS_DOWN | Previous link status is down. |
| ETHER_PREVIOUS_LINK_STATUS_UP | Previous link status is up. |

### ◆ ether_link_change_t

| enum ether_link_change_t | |
|---|---|
| Enumerator | |
| ETHER_LINK_CHANGE_NO_CHANGE | Link status is no change. |
| ETHER_LINK_CHANGE_LINK_DOWN | Link status changes to down. |
| ETHER_LINK_CHANGE_LINK_UP | Link status changes to up. |
| ETHER_LINK_CHANGE_NO_CHANGE | Link status is no change. |
| ETHER_LINK_CHANGE_LINK_DOWN | Link status changes to down. |
| ETHER_LINK_CHANGE_LINK_UP | Link status changes to up. |

### ◆ ether_magic_packet_t

| enum ether_magic_packet_t | |
|---|---|
| Enumerator | |
| ETHER_MAGIC_PACKET_NOT_DETECTED | Magic packet is not detected. |
| ETHER_MAGIC_PACKET_DETECTED | Magic packet is detected. |

### ◆ ether_link_establish_status_t

| enum ether_link_establish_status_t | |
|---|---|
| Enumerator | |
| ETHER_LINK_ESTABLISH_STATUS_DOWN | Link establish status is down. |
| ETHER_LINK_ESTABLISH_STATUS_UP | Link establish status is up. |
| ETHER_LINK_ESTABLISH_STATUS_DOWN | Link establish status is down. |
| ETHER_LINK_ESTABLISH_STATUS_UP | Link establish status is up. |

### ◆ ether_eesr_event_mask_t

| enum ether_eesr_event_mask_t |
|---|
| Event mask for EESR register |

| Enumerator | |
|---|---|
| ETHER_EESR_EVENT_MASK_CERF | CERF event. |
| ETHER_EESR_EVENT_MASK_PRE | PRE event. |
| ETHER_EESR_EVENT_MASK_RTSF | RTSF event. |
| ETHER_EESR_EVENT_MASK_RTLF | RTLF event. |
| ETHER_EESR_EVENT_MASK_RRF | PRF event. |
| ETHER_EESR_EVENT_MASK_RMAF | RMAF event. |
| ETHER_EESR_EVENT_MASK_TRO | TRO event. |
| ETHER_EESR_EVENT_MASK_CD | CD event. |
| ETHER_EESR_EVENT_MASK_DLC | DLC event. |
| ETHER_EESR_EVENT_MASK_CND | CND event. |
| ETHER_EESR_EVENT_MASK_RFOF | RFOF event. |
| ETHER_EESR_EVENT_MASK_RDE | RDE event. |
| ETHER_EESR_EVENT_MASK_FR | FR event. |
| ETHER_EESR_EVENT_MASK_TFUF | TFUF event. |
| ETHER_EESR_EVENT_MASK_TDE | TDE event. |
| ETHER_EESR_EVENT_MASK_TC | TC event. |
| ETHER_EESR_EVENT_MASK_ECI | ECI event. |
| ETHER_EESR_EVENT_MASK_ADE | ADE event. |
| ETHER_EESR_EVENT_MASK_RFCOF | RFCOF event. |
| ETHER_EESR_EVENT_MASK_RABT | RABT event. |
| ETHER_EESR_EVENT_MASK_TABT | TABT event. |
| ETHER_EESR_EVENT_MASK_TWB | TWB event. |

### ◆ ether_ecsr_event_mask_t

| enum ether_ecsr_event_mask_t | |
|---|---|
| Event mask for ECSR register | |
| Enumerator | |
| ETHER_ECSR_EVENT_MASK_ICD | ICD event. |
| ETHER_ECSR_EVENT_MASK_MPD | MPD event. |
| ETHER_ECSR_EVENT_MASK_LCHNG | LCHNG event. |
| ETHER_ECSR_EVENT_MASK_PSRTO | PSRTO event. |
| ETHER_ECSR_EVENT_MASK_BFR | BFR event. |

## Function Documentation

### ◆ R_ETHER_Open()

fsp_err_t R_ETHER_Open ( ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg* )

After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | Initialization of PHY-LSI failed. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to extend config structure or MAC address is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Interrupt is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

### ◆ R_ETHER_Close()

fsp_err_t R_ETHER_Close ( ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHER_BufferRelease()

fsp_err_t R_ETHER_BufferRelease ( ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |

### ◆ R_ETHER_RxBufferUpdate()

fsp_err_t R_ETHER_RxBufferUpdate ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer* )

Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | A pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | The pointer of buffer is NULL or not aligned on a 32-bit boundary. |
| FSP_ERR_INVALID_MODE | Driver is configured to non zero copy mode. |
| FSP_ERR_ETHER_RECEIVE_BUFFER_ACTIVE | All descriptor is active. |

### ◆ R_ETHER_LinkProcess()

fsp_err_t R_ETHER_LinkProcess ( ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Link is up. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Link is down. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | When reopening the PHY interface initialization of the PHY-LSI failed. |
| FSP_ERR_ALREADY_OPEN | When reopening the PHY interface it was already opened. |
| FSP_ERR_INVALID_CHANNEL | When reopening the PHY interface an invalid channel was passed. |
| FSP_ERR_INVALID_POINTER | When reopening the PHY interface the MAC address pointer was NULL. |
| FSP_ERR_INVALID_ARGUMENT | When reopening the PHY interface the interrupt was not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of the PHY-LSI failed. |

### ◆ R_ETHER_WakeOnLANEnable()

fsp_err_t R_ETHER_WakeOnLANEnable ( ether_ctrl_t *const *p_ctrl*)

The setting of ETHERC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

### ◆ R_ETHER_Read()

fsp_err_t R_ETHER_Read ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *length_bytes* )

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_NO_DATA | There is no data in receive buffer. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_FILTERING | Multicast Frame filter is enable, and Multicast Address Frame is received. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |

### ◆ R_ETHER_Write()

fsp_err_t R_ETHER_Write ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *frame_length* )

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE | As a Magic Packet is being detected, transmission and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | Transmit buffer is not empty. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Value of the send frame size is out of range. |

### ◆ R_ETHER_TxStatusGet()

fsp_err_t R_ETHER_TxStatusGet ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer_address* )

Provides status of Ethernet driver in the user provided pointer. Implements ether_api_t::txStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transmit buffer address is stored in provided p_buffer_address. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | p_status is NULL. |
| FSP_ERR_NOT_FOUND | Transmit buffer address has been overwritten in transmit descriptor. |

### ◆ R_ETHER_CallbackSet()

fsp_err_t R_ETHER_CallbackSet ( ether_ctrl_t *const *p_api_ctrl*, void(*)(ether_callback_args_t *) *p_callback*, void *const *p_context*, ether_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements ether_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

## 5.2.12.17 Ethernet (r_ether_phy)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_ETHER_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br>Resets Ethernet PHY device. Implements ether_phy_api_t::open. More... |
| fsp_err_t | R_ETHER_PHY_Close (ether_phy_ctrl_t *const p_ctrl)<br>Close Ethernet PHY device. Implements ether_phy_api_t::close. More... |
| fsp_err_t | R_ETHER_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl)<br>Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More... |
| fsp_err_t | R_ETHER_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)<br>Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More... |

| | |
|---|---|
| fsp_err_t | R_ETHER_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl)<br>Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More... |
| fsp_err_t | R_ETHER_PHY_ChipInit (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br>Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit. More... |
| fsp_err_t | R_ETHER_PHY_Read (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data)<br>Read data from register of PHY-LSI . Implements ether_phy_api_t::read. More... |
| fsp_err_t | R_ETHER_PHY_Write (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)<br>Write data to register of PHY-LSI . Implements ether_phy_api_t::write. More... |

### Detailed Description

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.

# Overview

The Ethernet PHY module is used to setup and manage an external Ethernet PHY device for use with the on-chip Ethernet Controller (ETHERC) peripheral. It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

### Features

The Ethernet PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

# Configuration

### Build Time Configurations for r_ether_phy

The following build time configurations are defined in fsp_cfg/r_ether_phy_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| KSZ8091RNB Target | • Disabled<br>• Enabled | Disabled | Select whether to use KSZ8091RNB PHY-LSI or not. |
| KSZ8041 Target | • Disabled<br>• Enabled | Disabled | Select whether to use KSZ8041 PHY-LSI or not. |
| DP83620 Target | • Disabled<br>• Enabled | Disabled | Select whether to use DP83620 PHY-LSI or not. |
| ICS1894 Target | • Disabled<br>• Enabled | Disabled | Select whether to use ICS1894 PHY-LSI or not. |
| User Own Target | • Disabled<br>• Enabled | Disabled | Select whether to use User own PHY-LSI or not. |
| Reference Clock | • Default<br>• Enabled<br>• Disabled | Default | Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board. |
| Reference Clock | • Default<br>• Enabled<br>• Disabled | Default | Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board. |
| Automatic PHY-LSI Initialization | • Enabled<br>• Disabled | Enabled | Select whether to initialize the Ethernet PHY-LSI in the open function. |

**Configurations for Networking > Ethernet (r_ether_phy)**

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ether_phy). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ether_phy0 | Module name. |

| | | | |
|---|---|---|---|
| Channel | • 0<br>• 1 | 0 | Select the Ethernet controller channel number. |
| PHY-LSI Address | Specify a value between 0 and 31. | 0 | Specify the address of the PHY-LSI used. |
| Default PHY-LSI port | 0 | 0 | Specify the default port for PHY-LSI configuration. |
| PHY-LSI Reset Completion Timeout | Specify a value between 0x1 and 0xFFFFFFFF. | 0x00020000 | Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used. |
| Select MII type | • MII<br>• RMII | RMII | Specify whether to use MII or RMII. |
| PHY-LSI type | • Kit Component<br>• DEFAULT<br>• KSZ8091RNB<br>• KSZ8041<br>• DP83620<br>• ICS1894<br>• User own PHY | Kit Component | Select the Phy LSI target. Selecting 'Kit Component' will automatically choose the correct option when using a Renesas development board. |
| Port Custom Init Function | Name must be a valid C symbol | NULL | Set the initial function of the PHY-LSI, When using your own PHY-LSI. |
| Port Custom Link Partner Ability Get Function | Name must be a valid C symbol | NULL | Set the link partner ability get function of the PHY-LSI, When using your own PHY-LSI. |
| MII/RMII Register Access Wait-time | Specify a value between 0x1 and 0x7FFFFFFF. | 8 | Specify the bit timing for MII/RMII register accesses during PHY initialization. This value should be adjusted experimentally based on the PHY-LSI used. |
| Flow Control | • Disable<br>• Enable | Disable | Select whether to enable or disable flow control. |

# Usage Notes

*Note*

*See the example below for details on how to initialize the Ethernet PHY module.*

### Accessing the MII and RMII Registers

Use the PIR register to access the MII and RMII registers in the PHY-LSI. Serial data in the MII and RMII management frame format is transmitted and received through the ET0_MDC and ET0_MDIO pins controlled by software.

### MII and RMII management frame format

The below table lists the MII and RMII management frame formats.

| Access type | MII and RMII management frame | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Item | PRE | ST | OP | PHYAD | REGAD | TA | DATA | IDLE |
| | Number of bits | 32 | 2 | 2 | 5 | 5 | 2 | 16 | 1 |
| Read | | 1...1 | 01 | 10 | 00001 | RRRRR | Z0 | DDDDDDDDDDDDDDDD | Z |
| Write | | 1...1 | 01 | 01 | 00001 | RRRRR | 10 | DDDDDDDDDDDDDDDD | Z |

*Note*

*- PRE (preamble): Send 32 consecutive 1s.*
*- ST (start of frame): Send 01b.*
*- OP (operation code): Send 10b for read or 01b for write.*
*- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.*
*- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.*
*- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.*
*Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).*
*(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)*
*- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.*
*- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write*
*operation (Z is output). No control is required, because a bus was already released during a read operation.*

### Limitations

- The r_ether_phy module may need to be customized for PHY devices other than the ones currently supported. Use the existing code as a starting point for creating a custom implementation. Supported PHY devices are as follows.
    - KSZ8091RNB
    - KSZ8041
    - DP83620
    - ICS1894

# Examples

### ETHER PHY Basic Example

This is a basic example of minimal use of the ETHER PHY in an application.

```
void ether_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ether_phy0_ctrl.open   = 0U;
    g_ether_phy0_cfg.channel = 0;
 /* Initializes the module. */
    err = R_ETHER_PHY_Open(&g_ether_phy0_ctrl, &g_ether_phy0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_ETHER_PHY_StartAutoNegotiate(&g_ether_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Polling until link is established. */
 while (FSP_SUCCESS != R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl))
    {
 /* Do nothing */
    }
 /* Get link partner ability from phy interface. */
    err = R_ETHER_PHY_LinkPartnerAbilityGet(&g_ether_phy0_ctrl,
                                            &g_ether_phy0_line_speed_duplex,
                                            &g_ether_phy0_local_pause,
                                            &g_ether_phy0_partner_pause);
    assert(FSP_SUCCESS == err);
 /* Check current link status. */
    err = R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

#### Data Structures

| | |
|---|---|
| struct | ether_phy_instance_ctrl_t |

| | |
|---|---|
| struct | ether_phy_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | ether_phy_interface_status_t |

## Data Structure Documentation

### ◆ ether_phy_instance_ctrl_t

| struct ether_phy_instance_ctrl_t | | |
|---|---|---|
| ETHER PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| ether_phy_cfg_t const * | p_ether_phy_cfg | Pointer to initial configurations. |
| volatile uint32_t * | p_reg_pir | Pointer to ETHERC peripheral registers. |
| uint32_t | local_advertise | Capabilities bitmap for local advertising. |
| ether_phy_interface_status_t | interface_status | Initialized status of ETHER PHY interface. |

### ◆ ether_phy_extended_cfg_t

| struct ether_phy_extended_cfg_t | |
|---|---|
| ETHER PHY extended configuration. | |
| **Data Fields** | |
| void(* | p_target_init )(ether_phy_instance_ctrl_t *p_instance_ctrl) |
| | Pointer to callback that is called to initialize the target. |
| | |
| bool(* | p_target_link_partner_ability_get )(ether_phy_instance_ctrl_t *p_instance_ctrl, uint32_t line_speed_duplex) |
| | Pointer to callback that is called to get the link partner ability. |
| | |
| ether_phy_lsi_cfg_t const * | p_phy_lsi_cfg_list [BSP_FEATURE_ETHER_NUM_CHANNELS] |
| | Pointer list of PHY LSI configurations. |
| | |
| uint8_t | default_phy_lsi_cfg_index |

| | |
|---|---|
| | Index of the default PHY LSI condiguration. |
| | |

## Enumeration Type Documentation

### ◆ ether_phy_interface_status_t

| enum ether_phy_interface_status_t | |
|---|---|
| Initialization state for read/write | |
| Enumerator | |
| ETHER_PHY_INTERFACE_STATUS_UNINITIALIZED | ETHER PHY interface is uninitialized. |
| ETHER_PHY_INTERFACE_STATUS_INITIALIZED | ETHER PHY interface is initialized. |

## Function Documentation

### ◆ R_ETHER_PHY_Open()

fsp_err_t R_ETHER_PHY_Open ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Resets Ethernet PHY device. Implements ether_phy_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |
| FSP_ERR_TIMEOUT | PHY-LSI Reset wait timeout. |
| FSP_ERR_INVALID_ARGUMENT | Register address is incorrect |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized. |

### ◆ R_ETHER_PHY_Close()

fsp_err_t R_ETHER_PHY_Close ( ether_phy_ctrl_t *const *p_ctrl*)

Close Ethernet PHY device. Implements ether_phy_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_ETHER_PHY_StartAutoNegotiate()

fsp_err_t R_ETHER_PHY_StartAutoNegotiate ( ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully starts auto-negotiate. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_INVALID_ARGUMENT | Register address is incorrect |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHER_PHY_LinkPartnerAbilityGet()

fsp_err_t R_ETHER_PHY_LinkPartnerAbilityGet ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause* )

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | The auto-negotiation isn't completed |
| FSP_ERR_INVALID_ARGUMENT | Status register address is incorrect |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHER_PHY_LinkStatusGet()

fsp_err_t R_ETHER_PHY_LinkStatusGet ( ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_INVALID_ARGUMENT | Status register address is incorrect |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHER_PHY_ChipInit()

fsp_err_t R_ETHER_PHY_ChipInit ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PHY device initialized successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block or configuration structure is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address or data is not a valid size. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_TIMEOUT | PHY-LSI Reset wait timeout. |

### ◆ R_ETHER_PHY_Read()

fsp_err_t R_ETHER_PHY_Read ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *const *p_data* )

Read data from register of PHY-LSI . Implements ether_phy_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully read data. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address is not a valid size |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

### ◆ R_ETHER_PHY_Write()

| fsp_err_t R_ETHER_PHY_Write ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *data* ) |
|---|
| Write data to register of PHY-LSI . Implements ether_phy_api_t::write. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ETHER_PHY successfully write data. |
| FSP_ERR_ASSERTION | Pointer to ETHER_PHY control block is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address or data is not a valid size |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |

## 5.2.12.18 Ethernet (r_rmac)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_RMAC_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)<br>After RMAC, Switch and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open. More... |
| fsp_err_t | R_RMAC_Close (ether_ctrl_t *const p_ctrl)<br>Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More... |
| fsp_err_t | R_RMAC_BufferRelease (ether_ctrl_t *const p_ctrl)<br>Move to the next buffer in the receive buffer list. Implements ether_api_t::bufferRelease. More... |
| fsp_err_t | R_RMAC_RxBufferUpdate (ether_ctrl_t *const p_ctrl, void *const p_buffer)<br>Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate. More... |
| fsp_err_t | R_RMAC_LinkProcess (ether_ctrl_t *const p_ctrl)<br>The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements |

| | |
|---|---|
| | ether_api_t::linkProcess. More... |
| fsp_err_t | R_RMAC_WakeOnLANEnable (ether_ctrl_t *const p_ctrl)<br>The setting of RMAC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable. More... |
| fsp_err_t | R_RMAC_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes)<br>Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read. More... |
| fsp_err_t | R_RMAC_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)<br>Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write. More... |
| fsp_err_t | R_RMAC_TxStatusGet (ether_ctrl_t *const p_ctrl, void *const p_buffer_address) |
| fsp_err_t | R_RMAC_CallbackSet (ether_ctrl_t *const p_api_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_RMAC_SetWriteConfig (ether_ctrl_t *const p_ctrl, rmac_write_cfg_t *const p_write_cfg)<br>Set configuration for tx frame. This API must call before ether_api_t::write. More... |
| fsp_err_t | R_RMAC_GetTxTimestamp (ether_ctrl_t *const p_ctrl, rmac_timestamp_t *const p_timestamp)<br>Get timestamp of transmitted frame. This API must call after ether_api_t::write. More... |
| fsp_err_t | R_RMAC_GetRxTimestamp (ether_ctrl_t *const p_ctrl, rmac_timestamp_t *const p_timestamp)<br>Get timestamp of received frame. This API must call before |

ether_api_t::read. More...

### Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface.

# Overview

This module transmits and receives Ethernet (IEEE 802.3) frames. This module provides single MAC layer interface.

The following figure shows which IPs are used by the r_rmac module.

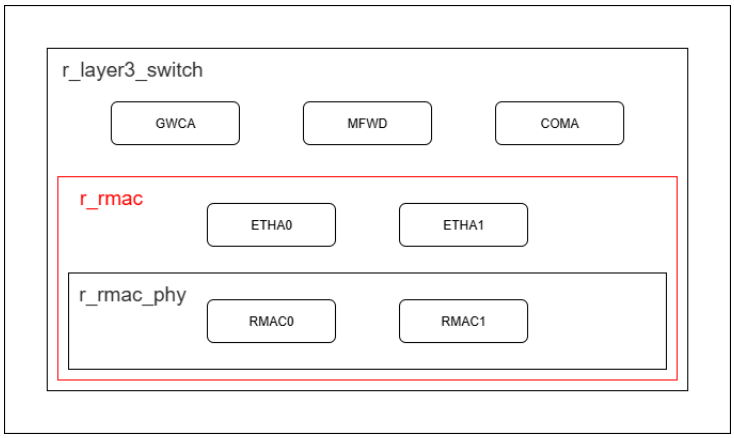



Figure 318: ESWM block diagram

### Features

The RMAC module supports the following features:

- Transmit/Receive processing
- Optional zero-copy buffering
- Callback function with returned event code
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support
- Get the frame transmission and reception times

# Configuration

### Build Time Configurations for r_rmac

The following build time configurations are defined in fsp_cfg/r_rmac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Networking > Ethernet MAC (r_rmac)**

This module can be added to the Stacks tab via New Stack > Networking > Ethernet MAC (r_rmac). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_ether0 | Module name. |
| Channel | Must be an integer greater than or equal to 0. | 0 | Write the ether channel number. |
| Zero-copy Mode | • Disable<br>• Enable | Disable | Enable or disable zero-copy mode. |
| Flow control functionality | • Disable<br>• Enable | Disable | Enable or disable flow control. |
| MAC address | Must be a valid MAC address | 00:11:22:33:44:55 | MAC address of this channel. |
| Filters | | | |
| Multicast Mode | • Disable<br>• Enable | Enable | Enable or disable multicast frame reception. |
| Promiscuous Mode | • Disable<br>• Enable | Disable | Enable this option to receive packets addressed to other NICs. |
| Broadcast filter | Must be a valid non-negative integer with maximum configurable value of 65535. | 0 | Limit of the number of broadcast frames received continuously |
| Buffers | | | |
| Padding size | Disable | Disable | The padding size that is automatically inserted into the received packets |
| Padding offset | Must be less than 64 bytes. | 0 | The offset into a receive buffer to insert padding bytes. |
| TX queue length | Value must be an integer | 3 | Number of buffers per TX queue. |

| | | | |
|---|---|---|---|
| RX queue length | Value must be an integer | 3 | Number of buffers per RX queue. |
| TX queue num | Value must be an integer | 2 | TX queue num |
| RX queue num | Value must be an integer | 2 | RX queue num |
| Number of TX buffer | Value must be an integer | 12 | Number of transmit buffers |
| Number of RX buffer | Value must be an integer | 12 | Number of receive buffers |
| Buffer size | Must be at least 1514 which is the maximum Ethernet frame size. | 1514 | Size of Ethernet buffer |
| Allocate RX buffer | • Disable<br>• Enable | Enable | Allocates the RX buffer when generating the configuration structure |
| Number of node num | Value must be an integer | 24 | Number of receive buffers |
| Interrupts | | | |
| Magic packet interrupt priority | MCU Specific Options | | Select the magic packet detection interrupt priority. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |
| Timestamp | | | |
| TS queue length | Value must be an integer | 8 | Timestamp queue length. |

### Interrupt Configuration

The first R_RMAC_Open function call sets GWDI interrupts. The user could provide callback function which would be invoked when GWDI interrupt handler has been completed. The callback arguments will contain information about a channel number, the event code and a pointer to the user defined context.

### Callback Configuration

The user could provide callback function which would be invoked when either a magic packet or a link signal change is detected. When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.4 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value should be copied into, for example, a global variable.

### Clock Configuration

The ESWM peripheral uses ICLK, PCLKA, ESWCLK and ESWPHYCLK as clocks. These clocks must be set to frequencies within the ranges shown in the following table.

| PHY speed [Mbps] | ICLK [MHz] | PCLKA [MHz] | ESWCLK [MHz] | ESWPHYCLK [MHz] |
|---|---|---|---|---|
| 10/100 | 50 to 250 | up to 125 | 50 to 250 | Not required |
| 1000 | 150 to 250 | up to 125 | 150 to 250 | 500 |

*Note*

1. *Clock restrictions are as follow: ICLK × 1.5 => ESWCLK, ESWCLK > PCLKA and ICLK > PCLKA.*
2. *When using a products in the HLQFP176 package, set ESWCLK to 125 MHz or less.*

### Pin Configuration

To use the RMAC module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). Please perform the pin setting before calling the R_RMAC_Open function.

# Usage Notes

### Ethernet Frame Format

The RMAC module supports the Ethernet II/IEEE 802.3 frame format.

### Frame Format for Data Transmission and Reception

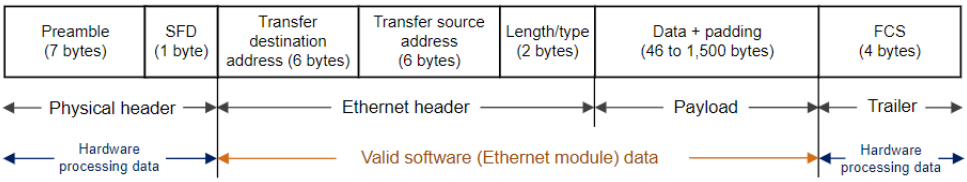



Figure 319: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

### PAUSE Frame Format

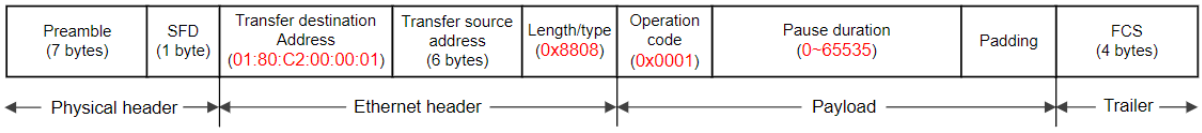



Figure 320: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and

the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

### Magic Packet Frame Format

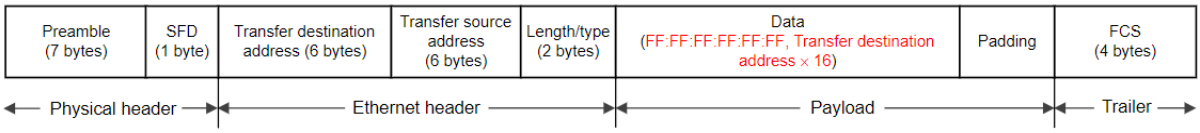



Figure 321: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

# Examples

### ETHER Basic Example

This is a basic example of minimal use of the ETHER in an application.

*Note*

*In this example zero-copy mode is disabled and there are no restrictions on buffer alignment.*

```
#define RMAC_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE (1514)
#define RMAC_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define RMAC_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define RMAC_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x02
#define RMAC_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define RMAC_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

/* Receive data buffer */
static uint8_t gp_read_buffer[RMAC_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE] = {0};
/* Transmit data buffer */
static uint8_t gp_send_data[RMAC_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    RMAC_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    RMAC_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
```

```
    RMAC_EXAMPLE_FRAME_TYPE,             /* Type field */
    RMAC_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void rmac_basic_example (void)
{
 fsp_err_t err            = FSP_SUCCESS;
    uint32_t read_data_size = 0;
 /* Open the rmac instance with initial configuration. */
    err = R_RMAC_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_RMAC_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_RMAC_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    assert(FSP_SUCCESS == err);
 /* Received data copy to user buffer from internal buffer. */
    err = R_RMAC_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the rmac instance. */
 R_RMAC_Close(&g_ether0_ctrl);
}
```

### Timestamp Example

This is a example of getting transmission and reception timestamps.

```
void rmac_timestamp_example (void)
{
```

```
fsp_err_t err            = FSP_SUCCESS;
   uint32_t read_data_size = 0;
rmac_timestamp_t rx_timestamp;
rmac_timestamp_t tx_timestamp;
rmac_write_cfg_t write_cfg = {.tx_timestamp_enable = 1};
/* Open the rmac instance with initial configuration. */
   err = R_RMAC_Open(&g_ether0_ctrl, &g_ether0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
do
   {
/* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
 * Initializes the module and make auto negotiation. */
      err = R_RMAC_LinkProcess(&g_ether0_ctrl);
   } while (FSP_SUCCESS != err);
/* Set write configuration. */
   err = R_RMAC_SetWriteConfig(&g_ether0_ctrl, &write_cfg);
   assert(FSP_SUCCESS == err);
/* Transmission is non-blocking. */
/* User data copy to internal buffer and is transferred by DMA in the background. */
   err = R_RMAC_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
   assert(FSP_SUCCESS == err);
/* Get TX timestamp value. */
   err = R_RMAC_GetTxTimestamp(&g_ether0_ctrl, &tx_timestamp);
   assert(FSP_SUCCESS == err);
/* Get RX timestamp value. */
   err = R_RMAC_GetRxTimestamp(&g_ether0_ctrl, &rx_timestamp);
   assert(FSP_SUCCESS == err);
/* Received data copy to user buffer from internal buffer. */
   err = R_RMAC_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
   assert(FSP_SUCCESS == err);
/* Disable transmission and receive function and close the rmac instance. */
R_RMAC_Close(&g_ether0_ctrl);
```

```
}
```

### Function Documentation

#### ◆ R_RMAC_Open()

fsp_err_t R_RMAC_Open ( ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg* )

After RMAC, Switch and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open.

Initializes the ether module and applies configurations. Implements ether_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | Initialization of PHY-LSI failed. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to extend config structure or MAC address is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Interrupt is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

#### ◆ R_RMAC_Close()

fsp_err_t R_RMAC_Close ( ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_RMAC_BufferRelease()

fsp_err_t R_RMAC_BufferRelease ( ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the receive buffer list. Implements ether_api_t::bufferRelease.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block or internal buffers is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_BUFFER_EMPTY | There is no available internal RX buffer. |

### ◆ R_RMAC_RxBufferUpdate()

fsp_err_t R_RMAC_RxBufferUpdate ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer* )

Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | A pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | The pointer of buffer is NULL or not aligned on a 32-bit boundary. |
| FSP_ERR_INVALID_MODE | Driver is configured to non zero copy mode. |
| FSP_ERR_BUFFER_EMPTY | There is no available internal RX buffer. |

### ◆ R_RMAC_LinkProcess()

fsp_err_t R_RMAC_LinkProcess ( ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Link is up. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Link is down. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | When reopening the PHY interface initialization of the PHY-LSI failed. |
| FSP_ERR_ALREADY_OPEN | When reopening the PHY interface it was already opened. |
| FSP_ERR_INVALID_CHANNEL | When reopening the PHY interface an invalid channel was passed. |
| FSP_ERR_INVALID_POINTER | When reopening the PHY interface the MAC address pointer was NULL. |
| FSP_ERR_INVALID_ARGUMENT | When reopening the PHY interface the interrupt was not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of the PHY-LSI failed. |

### ◆ R_RMAC_WakeOnLANEnable()

fsp_err_t R_RMAC_WakeOnLANEnable ( ether_ctrl_t *const *p_ctrl*)

The setting of RMAC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_PHY_ERROR_LINK | Initialization of PHY-LSI failed. |

### ◆ R_RMAC_Read()

fsp_err_t R_RMAC_Read ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *length_bytes* )

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_NO_DATA | There is no data in receive buffer. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_FILTERING | Multicast Frame filter is enable, and Multicast Address Frame is received. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |
| FSP_ERR_BUFFER_EMPTY | There is no available internal RX buffer. |

### ◆ R_RMAC_Write()

fsp_err_t R_RMAC_Write ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *frame_length* )

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_ETHER_ERROR_LINK | Auto-negotiation is not completed, and reception is not enabled. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | Transmit buffer is not empty. |
| FSP_ERR_INVALID_POINTER | Value of the pointer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Value of the send frame size is out of range. |
| FSP_ERR_BUFFER_EMPTY | There is no available internal TX buffer. |

### ◆ R_RMAC_TxStatusGet()

fsp_err_t R_RMAC_TxStatusGet ( ether_ctrl_t *const *p_ctrl*, void *const *p_buffer_address* )

Provides status of Ethernet driver in the user provided pointer. Implements ether_api_t::txStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transmit buffer address is stored in provided p_buffer_address. |
| FSP_ERR_ASSERTION | Pointer to ETHER control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | p_status is NULL. |
| FSP_ERR_NOT_FOUND | Transmit buffer address has been overwritten in transmit descriptor. |

### ◆ R_RMAC_CallbackSet()

fsp_err_t R_RMAC_CallbackSet ( ether_ctrl_t *const *p_api_ctrl*, void(*)(ether_callback_args_t *) *p_callback*, void *const *p_context*, ether_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements ether_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_RMAC_SetWriteConfig()

fsp_err_t R_RMAC_SetWriteConfig ( ether_ctrl_t *const *p_ctrl*, rmac_write_cfg_t *const *p_write_cfg* )

Set configuration for tx frame. This API must call before ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | A pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_RMAC_GetTxTimestamp()

fsp_err_t R_RMAC_GetTxTimestamp ( ether_ctrl_t *const *p_ctrl*, rmac_timestamp_t *const *p_timestamp* )

Get timestamp of transmitted frame. This API must call after ether_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | A pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_RMAC_GetRxTimestamp()

fsp_err_t R_RMAC_GetRxTimestamp ( ether_ctrl_t *const *p_ctrl*, rmac_timestamp_t *const *p_timestamp* )

Get timestamp of received frame. This API must call before ether_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Processing completed successfully. |
| FSP_ERR_ASSERTION | A pointer argument is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

## 5.2.12.19 Ethernet (r_rmac_phy)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_RMAC_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br>Resets Ethernet PHY device. Implements ether_phy_api_t::open. More... |
| fsp_err_t | R_RMAC_PHY_Close (ether_phy_ctrl_t *const p_ctrl)<br>Close Ethernet PHY device. Implements ether_phy_api_t::close. More... |
| fsp_err_t | R_RMAC_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl)<br>Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More... |
| fsp_err_t | R_RMAC_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)<br>Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More... |
| fsp_err_t | R_RMAC_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl)<br>Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More... |

| | |
|---|---|
| fsp_err_t | R_RMAC_PHY_ChipInit (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)<br>Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit. More... |
| fsp_err_t | R_RMAC_PHY_Read (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data)<br>Read data from register of PHY-LSI . Implements ether_phy_api_t::read. More... |
| fsp_err_t | R_RMAC_PHY_Write (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)<br>Write data to register of PHY-LSI . Implements ether_phy_api_t::write. More... |
| fsp_err_t | R_RMAC_PHY_ChipSelect (ether_phy_ctrl_t *const p_ctrl, uint8_t port)<br>Update the target PHY LSI of this driver. More... |

### Detailed Description

The RMAC PHY module (r_rmac_phy) provides an API for standard Ethernet PHY communications applications that use the RMAC peripheral. It implements the Ethernet PHY Interface.

# Overview

The RMAC PHY module is used to setup and manage an external Ethernet PHY device. r_rmac_phy use the on-chip MAC layer peripheral(RMAC), which is part of Ethernet Switch Module(ESWM). It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

The following figure shows which IPs are used by the r_rmac_phy module.

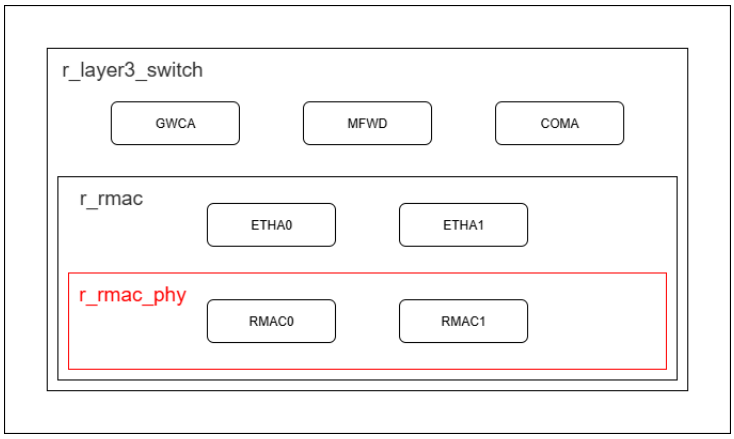



Figure 322: ESWM block diagram

### Features

The RMAC PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support
- Gigabit Ethernet support

Following features are not supported:

- Extensions to management frame format defined in IEEE802.3 clause 45

# Configuration

### Build Time Configurations for r_rmac_phy

The following build time configurations are defined in fsp_cfg/r_rmac_phy_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| KSZ8091RNB Target | • Disabled<br>• Enabled | Disabled | Select whether to use KSZ8091RNB PHY-LSI or not. |
| KSZ8041 Target | • Disabled<br>• Enabled | Disabled | Select whether to use KSZ8041 PHY-LSI or not. |
| DP83620 Target | • Disabled<br>• Enabled | Disabled | Select whether to use DP83620 PHY-LSI or not. |
| ICS1894 Target | • Disabled<br>• Enabled | Disabled | Select whether to use ICS1894 PHY-LSI or not. |

| | | | |
|---|---|---|---|
| GPY111 Target | • Disabled<br>• Enabled | Disabled | Select whether to use GPY111 PHY-LSI or not. |
| User Own Target | • Disabled<br>• Enabled | Disabled | Select whether to use User own PHY-LSI or not. |
| Reference Clock | • Default<br>• Enabled<br>• Disabled | Default | Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board. |
| Reference Clock | • Default<br>• Enabled<br>• Disabled | Default | Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board. |

**Configurations for Networking > Ethernet (r_rmac_phy)**

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_rmac_phy). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rmac_phy0 | Module name. |
| Channel | • 0<br>• 1 | 0 | Select the Ethernet controller channel number. |
| Default PHY-LSI port | • 0<br>• 1 | 0 | Specify the default port for PHY-LSI configuration. |
| PHY-LSI Reset Completion Timeout | Specify a value between 0x1 and 0xFFFFFFFF. | 0x00020000 | Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used. |
| Select MII type | • MII<br>• RMII<br>• GMII<br>• RGMII | RMII | Specify whether to use MII or RMII. |

| | | | |
|---|---|---|---|
| Port Custom Init Function | Name must be a valid C symbol | NULL | Set the initial function of the PHY-LSI, When using your own PHY-LSI. |
| Port Custom Link Partner Ability Get Function | Name must be a valid C symbol | NULL | Set the link partner ability get function of the PHY-LSI, When using your own PHY-LSI. |
| Flow Control | • Disable<br>• Enable | Disable | Select whether to enable or disable flow control. |
| Management frame format | Clause 22 frame format | Clause 22 frame format | Select the management frame format used to access the PHY-LSI. |
| MDC clock rate (Hz) | Value must be a non-negative number | 2500000 | Specify the MDC clock frequency |
| MDIO hold timing adjustment | Value must be a non-negative integer less than 8 | 0 | Specify the clock cycle for hold timing. |
| MDIO capture timing adjustment | Value must be a non-negative integer less than 8 | 0 | Specify the clock cycle for capture timing. |

# Usage Notes

### Accessing the Registers in PHY-LSI

Use the RMAC peripheral to access the registers in the PHY-LSI. Management frame is transmitted and received through the ETn_MDC and ETn_MDIO pins controlled by hardware.

This HW supports two management frame format.

- Normal management frame format defined in Clause 22 of IEEE802.3 (Describe "MDIO" in this document)
- Extension management frame format defined in Clause 45 of IEEE802.3 (Describe "eMDIO" in this document)

### Management frame format of MDIO (Clause 22)

The below table lists the management frame formats of MDIO.

| Access type | MII and RMII management frame | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item | PRE | ST | OP | PHYAD | REGAD | TA | DATA | IDLE |
| Number of bits | 32 | 2 | 2 | 5 | 5 | 2 | 16 | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Read | 1...1 | 01 | 10 | 00001 | RRRRR | Z0 | DDDDD DDDDD DDDDD D | Z |
| Write | 1...1 | 01 | 01 | 00001 | RRRRR | 10 | DDDDD DDDDD DDDDD D | Z |

*Note*

*- PRE (preamble): Send 32 consecutive 1s.*
*- ST (start of frame): Send 01b.*
*- OP (operation code): Send 10b for read or 01b for write.*
*- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.*
*- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.*
*- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.*
*Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).*
*(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)*
*- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.*
*- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write*
*operation (Z is output). No control is required, because a bus was already released during a read operation.*

### Management frame format of eMDIO (Clause 45)

The below table lists the management frame formats of eMDIO(Clause 45).

| Access type | MII and RMII management frame | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item | PRE | ST | OP | PRTAD | DEVAD | TA | DATA | IDLE |
| Number of bits | 32 | 2 | 2 | 5 | 5 | 2 | 16 | 1 |
| Address | 1...1 | 00 | 00 | PPPPP | EEEEE | 10 | AAAAAA AAAAAA AAAA | Z |
| Write | 1...1 | 00 | 01 | PPPPP | EEEEE | 10 | DDDDD DDDDD DDDDD D | Z |
| Read | 1...1 | 00 | 11 | PPPPP | EEEEE | Z0 | DDDDD DDDDD DDDDD D | Z |
| Read Increment | 1...1 | 00 | 10 | PPPPP | EEEEE | Z0 | DDDDD DDDDD DDDDD D | Z |

*Note*

*- PRE (preamble): Send 32 consecutive 1s.*
*- ST (start of frame): Send 00b.*
*- OP (operation code): Send 00b for register address, 10b for read operation, 11b for write operation, or 10b for post-read-increment-address operation.*
*- PRTAD (port address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits port address. When the port address is 1, send 00001b.*
*- DEVAD (device address): Up to 32 MDIO Manageable Devices(MMD) are contained in a PHY-LSI. MMD are selected with these 5 bits device address.*
*- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.*
*Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).*
*(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)*
*- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.*
*- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write*
*operation (Z is output). No control is required, because a bus was already released during a read operation.*

### Limitations

- The r_rmac_phy module may need to be customized for PHY devices other than the ones currently supported. Use the existing code as a starting point for creating a custom implementation. Supported PHY devices are as follows.
  - KSZ8091RNB
  - KSZ8041
  - DP83620
  - ICS1894
  - GPY111

# Examples

### RMAC PHY Basic Example

This is a basic example of minimal use of the RMAC PHY in an application.

```
void rmac_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_rmac_phy0_ctrl.open   = 0U;
    g_rmac_phy0_cfg.channel = 0;
 /* Initializes the module. */
    err = R_RMAC_PHY_Open(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
```

```
/* Polling until link is established. */
while (FSP_SUCCESS != R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl))
   {
/* Do nothing */
   }
/* Get link partner ability from phy interface. */
   err = R_RMAC_PHY_LinkPartnerAbilityGet(&g_rmac_phy0_ctrl,
                                          &g_rmac_phy0_line_speed_duplex,
                                          &g_rmac_phy0_local_pause,
                                          &g_rmac_phy0_partner_pause);
   assert(FSP_SUCCESS == err);
/* Check current link status. */
   err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
   assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | rmac_phy_instance_ctrl_t |
| struct | rmac_phy_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | rmac_phy_interface_status_t |
| enum | rmac_phy_frame_format_t |

## Data Structure Documentation

### ◆ rmac_phy_instance_ctrl_t

| struct rmac_phy_instance_ctrl_t | | |
|---|---|---|
| RMAC PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called. | | |
| Data Fields | | |
| uint32_t | open | Used to determine if the channel is configured. |
| ether_phy_cfg_t const * | p_ether_phy_cfg | Pointer to initial configurations. |
| R_RMAC0_Type * | p_reg_rmac | Pointer to RMAC peripheral registers. |
| | | |

| | | |
|---|---|---|
| uint32_t | local_advertise | Capabilities bitmap for local advertising. |
| rmac_phy_interface_status_t | interface_status | Initialized status of ETHER PHY interface. |
| uint8_t | phy_lsi_cfg_index | Index of the PHY LSI that is currently the target of operation. |

### ◆ rmac_phy_extended_cfg_t

| struct rmac_phy_extended_cfg_t | |
|---|---|
| RMAC PHY extended configuration. | |
| **Data Fields** | |
| void(* | p_target_init )(rmac_phy_instance_ctrl_t *p_instance_ctrl) |
| | Pointer to callback that is called to initialize the target. |
| | |
| bool(* | p_target_link_partner_ability_get )(rmac_phy_instance_ctrl_t *p_instance_ctrl, uint32_t line_speed_duplex) |
| | Pointer to callback that is called to get the link partner ability. |
| | |
| rmac_phy_frame_format_t | frame_format |
| | Whether the management frame format is MDIO or eMDIO. |
| | |
| uint32_t | mdc_clock_rate |
| | MDC frequency division. |
| | |
| uint8_t | mdio_hold_time |
| | MDIO hold time adjustment. |
| | |
| uint8_t | mdio_capture_time |
| | MDIO capture time adjustment. |
| | |
| ether_phy_lsi_cfg_t const * | p_phy_lsi_cfg_list [BSP_FEATURE_ETHER_NUM_CHANNELS] |
| | Pointer list of PHY LSI configurations. |

| | |
|---|---|
| uint8_t | default_phy_lsi_cfg_index |
| | Index of the default PHY LSI condiguration. |

## Enumeration Type Documentation

### ◆ rmac_phy_interface_status_t

| enum rmac_phy_interface_status_t | |
|---|---|
| Initialization state for read/write | |
| Enumerator | |
| RMAC_PHY_INTERFACE_STATUS_UNINITIALIZED | ETHER PHY interface is uninitialized. |
| RMAC_PHY_INTERFACE_STATUS_INITIALIZED | ETHER PHY interface is initialized. |

### ◆ rmac_phy_frame_format_t

| enum rmac_phy_frame_format_t | |
|---|---|
| Enumerator | |
| RMAC_PHY_FRAME_FORMAT_MDIO | Normal management frame format defined in clause 22. |
| RMAC_PHY_FRAME_FORMAT_EMDIO | Extension management frame format defined in clause 45. |

## Function Documentation

### ◆ R_RMAC_PHY_Open()

fsp_err_t R_RMAC_PHY_Open ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Resets Ethernet PHY device. Implements ether_phy_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_CHANNEL | Invalid channel number is given. |
| FSP_ERR_INVALID_POINTER | Pointer to p_cfg is NULL. |
| FSP_ERR_INVALID_MODE | Function is called when not in CONFIG mode. |

### ◆ R_RMAC_PHY_Close()

fsp_err_t R_RMAC_PHY_Close ( ether_phy_ctrl_t *const *p_ctrl*)

Close Ethernet PHY device. Implements ether_phy_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_INVALID_MODE | Function is called when not in DISABLE mode. |

### ◆ R_RMAC_PHY_StartAutoNegotiate()

fsp_err_t R_RMAC_PHY_StartAutoNegotiate ( ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RMAC_PHY successfully starts auto-negotiate. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_LinkPartnerAbilityGet()

fsp_err_t R_RMAC_PHY_LinkPartnerAbilityGet ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause* )

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RMAC_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_ETHER_PHY_NOT_READY | The auto-negotiation isn't completed |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_LinkStatusGet()

fsp_err_t R_RMAC_PHY_LinkStatusGet ( ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RMAC_PHY successfully get link partner ability. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_ETHER_PHY_ERROR_LINK | PHY-LSI is not link up. |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_ChipInit()

fsp_err_t R_RMAC_PHY_ChipInit ( ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg* )

Initialize Ethernet PHY device. Implements ether_phy_api_t::chipInit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PHY device initialized successfully. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_TIMEOUT | PHY-LSI Reset wait timeout. |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_Read()

fsp_err_t R_RMAC_PHY_Read ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *const *p_data* )

Read data from register of PHY-LSI . Implements ether_phy_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RMAC_PHY successfully read data. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_INVALID_POINTER | Pointer to read buffer is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address is not a valid size |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_Write()

fsp_err_t R_RMAC_PHY_Write ( ether_phy_ctrl_t *const *p_ctrl*, uint32_t *reg_addr*, uint32_t *data* )

Write data to register of PHY-LSI . Implements ether_phy_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RMAC_PHY successfully write data. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Address or data is not a valid size |
| FSP_ERR_NOT_INITIALIZED | The control block has not been initialized |
| FSP_ERR_INVALID_MODE | Function is called when not in OPERATION mode. |

### ◆ R_RMAC_PHY_ChipSelect()

| fsp_err_t R_RMAC_PHY_ChipSelect ( ether_phy_ctrl_t *const *p_ctrl*, uint8_t *port* ) |
|---|

Update the target PHY LSI of this driver.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | PHY device initialized successfully. |
| FSP_ERR_ASSERTION | Pointer to RMAC_PHY control block or configuration structure is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid PHY LSI is selected. |

## 5.2.12.20 Ethernet Generic PTP Timer (r_gptp)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | R_GPTP_Open (gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg) |
| fsp_err_t | R_GPTP_Close (gptp_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPTP_TimerCfg (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg) |
| fsp_err_t | R_GPTP_Start (gptp_ctrl_t *const p_ctrl, uint8_t timer) |
| fsp_err_t | R_GPTP_Stop (gptp_ctrl_t *const p_ctrl, uint8_t timer) |
| fsp_err_t | R_GPTP_TimerValueGet (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value) |
| fsp_err_t | R_GPTP_TimerOffsetSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset) |
| fsp_err_t | R_GPTP_TimerRateSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate) |
| fsp_err_t | R_GPTP_PulseGeneratorSet (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_pulse_generator_t *p_pulse) |
| fsp_err_t | R_GPTP_CallbackSet (gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, |

gptp_callback_args_t *const p_callback_memory)

### Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the GPTP Interface.

# Overview

Ethernet Generic PTP timer (GPTP) enables accurate synchronization of the clock in the control system.

This module supports two GPTP timer.

### Features

The GPTP module has the following features:

- Supports start and stop gPTP timer.
- Supports set timer offset.
- Supports set timer rate.
- Supports set pulse generator.

# Configuration

### Build Time Configurations for r_gptp

The following build time configurations are defined in fsp_cfg/r_gptp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Networking > Ethernet Generic PTP Timer (r_gptp)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet Generic PTP Timer (r_gptp). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_gptp0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |

### Clock Configuration

The GPTP clock is based on the ESWCLK frequency. You can set the PTPTIVCt (t = 0 or 1) register to

timer increment value according to ESWCLK frequency.

# Usage Notes

# Examples

This is a basic example of minimal use of the GPTP in an application.

```
void gptp_basic_example (void)
{
 fsp_err_t       err       = FSP_SUCCESS;
    uint8_t          timer    = GPTP_EXAMPLE_TIMER_NUMBER;
 gptp_timer_cfg_t timer_cfg = {.clock_period = GPTP_EXAMPLE_CLOCK_PERIOD};
    int64_t          offset    = GPTP_EXAMPLE_OFFSET;
    uint32_t         rate      = GPTP_EXAMPLE_RATE;
 gptp_timer_value_t timer_value =
    {
       .time_sec_upper = 0,
       .time_sec_lower = 0,
       .time_nsec      = 0,
    };
 gptp_pulsed_output_cfg_t pulsed_output_cfg0 =
    {
       .pulse_num       = 0,
       .start_sec_upper = 0,
       .start_sec_lower = 0,
       .start_ns         = 0,
       .wide             = 0,
       .period_sec_upper = 0,
       .period_sec_lower = 0,
       .period_ns       = 0,
    };
 gptp_pulsed_output_cfg_t pulsed_output_cfg1 =
    {
       .pulse_num       = 1,
       .start_sec_upper = 0,
```

```
      .start_sec_lower = 0,
      .start_ns         = 0,
      .wide             = 0,
      .period_sec_upper = 0,
      .period_sec_lower = 0,
      .period_ns       = 0,
   };
gptp_pulsed_output_cfg_t pulsed_output_cfg2 =
   {
      .pulse_num       = 2,
      .start_sec_upper = 0,
      .start_sec_lower = 0,
      .start_ns         = 0,
      .wide             = 0,
      .period_sec_upper = 0,
      .period_sec_lower = 0,
      .period_ns       = 0,
   };
gptp_pulsed_output_cfg_t pulsed_output_cfg3 =
   {
      .pulse_num       = 3,
      .start_sec_upper = 0,
      .start_sec_lower = 0,
      .start_ns         = 0,
      .wide             = 0,
      .period_sec_upper = 0,
      .period_sec_lower = 0,
      .period_ns       = 0,
   };
gptp_pulse_generator_t pulse_generator =
   {
      .p_pulsed_output_cfg_list =
     {&pulsed_output_cfg0, &pulsed_output_cfg1, &pulsed_output_cfg2,
&pulsed_output_cfg3},
```

```
    };
/* Open the gptp instance with initial configuration. */
    err = R_GPTP_Open(&g_gptp0_ctrl, &g_gptp0_cfg);
/* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
/* Set configuration of gptp timer. */
    err = R_GPTP_TimerCfg(&g_gptp0_ctrl, timer, &timer_cfg);
    assert(FSP_SUCCESS == err);
/* Start gptp timer. */
    err = R_GPTP_Start(&g_gptp0_ctrl, timer);
    assert(FSP_SUCCESS == err);
/* Get value of gptp timer. */
    err = R_GPTP_TimerValueGet(&g_gptp0_ctrl, timer, &timer_value);
    assert(FSP_SUCCESS == err);
/* Set offset of gptp timer. */
    err = R_GPTP_TimerOffsetSet(&g_gptp0_ctrl, timer, offset);
    assert(FSP_SUCCESS == err);
/* Set rate of gptp timer. */
    err = R_GPTP_TimerRateSet(&g_gptp0_ctrl, timer, rate);
    assert(FSP_SUCCESS == err);
/* Set pulse generator of gptp timer. */
    err = R_GPTP_PulseGeneratorSet(&g_gptp0_ctrl, timer, &pulse_generator);
    assert(FSP_SUCCESS == err);
/* Stop gptp timer. */
    err = R_GPTP_Stop(&g_gptp0_ctrl, timer);
    assert(FSP_SUCCESS == err);
/* Close the gptp instance. */
    err = R_GPTP_Close(&g_gptp0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | gptp_pulsed_output_cfg_t |
| struct | gptp_pulse_generator_t |

| | |
|---|---|
| struct | gptp_instance_ctrl_t |

## Data Structure Documentation

### ◆ gptp_pulsed_output_cfg_t

| struct gptp_pulsed_output_cfg_t | | |
|---|---|---|
| Pulsed output configuration. | | |
| Data Fields | | |
| uint8_t | pulse_num | Pulse generator number. |
| uint16_t | start_sec_upper | Pulse start second(upper 16 bit). |
| uint32_t | start_sec_lower | Pulse start second(lower 32 bit). |
| uint32_t | start_ns | Pulse start nanosecond. |
| uint16_t | wide | Pulse width. |
| uint16_t | period_sec_upper | Pulse period second(upper 16 bit). |
| uint32_t | period_sec_lower | Pulse period second(lower 32 bit). |
| uint32_t | period_ns | Pulse period nanosecond. |

### ◆ gptp_pulse_generator_t

| struct gptp_pulse_generator_t | | |
|---|---|---|
| Pulse generator. | | |
| Data Fields | | |
| gptp_pulsed_output_cfg_t * | p_pulsed_output_cfg_list[BSP_FEATURE_ESWM_GPTP_PULSE_GENERATOR_NUM] | List of pointer to pulsed output configuration. |

### ◆ gptp_instance_ctrl_t

| struct gptp_instance_ctrl_t | |
|---|---|
| GPTP control block. DO NOT INITIALIZE. Initialization occurs when gptp_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Indicates whether the open() API has been successfully called. |
| | |
| gptp_cfg_t const * | p_cfg |
| | Pointer to initial configurations. |

| | |
|---|---|
| | |
| R_GPTP_Type * | p_reg_gptp |
| | Pointer to GPTP timer register. |
| | |
| void(* | p_callback )(gptp_callback_args_t *) |
| | Pointer to callback that is called. |
| | |
| gptp_callback_args_t * | p_callback_memory |
| | Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory. |
| | |
| void const * | p_context |
| | Pointer to context to be passed into callback function. |
| | |

### Function Documentation

#### ◆ R_GPTP_Open()

fsp_err_t R_GPTP_Open ( gptp_ctrl_t *const *p_ctrl*, gptp_cfg_t const *const *p_cfg* )

Initializes the gptp module and applies configurations.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module opened successfully. |
| FSP_ERR_ASSERTION | Pointer to GPTP control block or configuration structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |

### ◆ R_GPTP_Close()

fsp_err_t R_GPTP_Close ( gptp_ctrl_t *const *p_ctrl*)

Close the gptp module.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Pointer to GPTP control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_GPTP_TimerCfg()

fsp_err_t R_GPTP_TimerCfg ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_timer_cfg_t const *const *p_timer_cfg* )

Configures the gptp timer parameters.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Command successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid timer id. |
| FSP_ERR_INVALID_POINTER | Invalid poiter to the timer cfg. |

### ◆ R_GPTP_Start()

fsp_err_t R_GPTP_Start ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer* )

Starts gptp timer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer started.. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid timer id. |

### ◆ R_GPTP_Stop()

fsp_err_t R_GPTP_Stop ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer* )

Stops gptp timer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer stopped. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid timer id. |

### ◆ R_GPTP_TimerValueGet()

fsp_err_t R_GPTP_TimerValueGet ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_timer_value_t *const *p_timer_value* )

Gets the current time value to the timer with the specified gptp timer number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Sec upper, sec lower, and nano sec srored in p_timestamp. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

### ◆ R_GPTP_TimerOffsetSet()

fsp_err_t R_GPTP_TimerOffsetSet ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, int64_t *offset* )

Sets the offset correction for the specified gptp timer number.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Time offset set successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

### ◆ R_GPTP_TimerRateSet()

fsp_err_t R_GPTP_TimerRateSet ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, uint32_t *rate* )

Sets clock rate correction for the specified timer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Time rate set successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

### ◆ R_GPTP_PulseGeneratorSet()

fsp_err_t R_GPTP_PulseGeneratorSet ( gptp_ctrl_t *const *p_ctrl*, uint8_t *timer*, gptp_pulse_generator_t * *p_pulse* )

Sets pulse generator.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pulse generator set successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to arguments are NULL. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameter. |

◆ **R_GPTP_CallbackSet()**

| fsp_err_t R_GPTP_CallbackSet ( gptp_ctrl_t *const *p_ctrl*, void(*)(gptp_callback_args_t *) *p_callback*, void *const *p_context*, gptp_callback_args_t *const *p_callback_memory* ) |
|---|
| Updates the user callback with the option to provide memory for the callback argument structure. Implements gptp_api_t::callbackSet. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### 5.2.12.21 FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp)

Modules » Networking

Middleware for using TCP on RA MCUs.

# Overview

FreeRTOS Plus TCP is a TCP stack created for use with FreeRTOS.

This module provides the NetworkInterface required to use FreeRTOS Plus TCP with the Ethernet (r_ether) driver or the Ethernet (r_rmac) driver.

Please refer to the FreeRTOS Plus TCP documentation for further details.

# Configuration

### Build Time Configurations for FreeRTOS_Plus_TCP

The following build time configurations are defined in aws/FreeRTOSIPConfig.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Print debug messages | • Disable<br>• Enable | Disable | If ipconfigHAS_DEBUG_PRINTF is set to 1 then FreeRTOS_debug_printf should be defined to the function used to print out the debugging |

| | | | |
|---|---|---|---|
| | | | messages. |
| Backward Compatible Mode | • No<br>• Yes | Yes | Run the code in backward compatible mode |
| Enable IPV6 | • Disable<br>• Enable | Disable | Stack supports handling IPv6 packets (including handling IPv6 header, ND, RA, and so on) when enabled |
| Print info messages | • Disable<br>• Enable | Disable | Set to 1 to print out non debugging messages, for example the output of the FreeRTOS_netstat() command, and ping replies. If ipconfigHAS_PRINTF is set to 1 then FreeRTOS_printf should be set to the function used to print out the messages. |
| Byte order of the target MCU | pdFREERTOS_LITTLE_ENDIAN | pdFREERTOS_LITTLE_ENDIAN | Define the byte order of the target MCU |
| IP/TCP/UDP checksums | • Disable<br>• Enable | Enable | If the network card/driver includes checksum offloading (IP/TCP/UDP checksums) then set ipconfigDRIVER_INCLUDED_RX_IP_CHECKSUM to 1 to prevent the software stack repeating the checksum calculations. |
| Receive Block Time | Value must be a non-negative integer | 10000 | Amount of time FreeRTOS_recv() will block for. The timeouts can be set per socket, using setsockopt(). |
| Send Block Time | Value must be a non-negative integer | 10000 | Amount of time FreeRTOS_send() will block for. The timeouts can be set per socket, using setsockopt(). |
| DNS caching | • Disable<br>• Enable | Enable | DNS caching |
| DNS Request Attempts | Value must be an | 2 | When a cache is |

| | | | |
|---|---|---|---|
| | integer | | present, ipconfigDNS_REQUEST_ATTEMPTS can be kept low and also DNS may use small timeouts. |
| IP stack task priority | Only symbols, numbers and arithmetic operators are valid. | configMAX_PRIORITIES - 2 | Set the priority of the task that executes the IP stack. |
| Stack size in words (not bytes) | Only symbols, numbers and arithmetic operators are valid. | configMINIMAL_STACK_SIZE * 5 | The size, in words (not bytes), of the stack allocated to the FreeRTOS+TCP stack. |
| Network Events call vApplicationIPNetworkEventHook | • Disable<br>• Enable | Enable | vApplicationIPNetworkEventHook is called when the network connects or disconnects. |
| Max UDP send block time | Only symbols, numbers and arithmetic operators are valid. | 15000 / portTICK_PERIOD_MS | Max UDP send block time |
| Use DHCP | • Disable<br>• Enable | Enable | If ipconfigUSE_DHCP is 1 then FreeRTOS+TCP will attempt to retrieve an IP address, netmask, DNS server address and gateway address from a DHCP server. |
| DHCP Register Hostname | • Disable<br>• Enable | Enable | Register hostname when using DHCP |
| DHCP Uses Unicast | • Disable<br>• Enable | Enable | DHCP uses unicast. |
| DHCP callback function | • Disable<br>• Enable | Disable | Provide an implementation of the DHCP callback function (xApplicationDHCPHook) |
| Interval between transmissions | Only symbols, numbers and arithmetic operators are valid. | 120000 / portTICK_PERIOD_MS | When ipconfigUSE_DHCP is set to 1, DHCP requests will be sent out at increasing time intervals until either a reply is received from a DHCP server and accepted, or the interval between transmissions reaches ipconfigMAXIMUM_DISC |

| | | | |
|---|---|---|---|
| | | | OVER_TX_PERIOD. |
| ARP Cache Entries | Value must be an integer | 6 | The maximum number of entries that can exist in the ARP table at any one time |
| ARP Request Retransmissions | Value must be an integer | 5 | ARP requests that do not result in an ARP response will be re-transmitted a maximum of ipconfigMAX_ARP_RETRANSMISSIONS times before the ARP request is aborted. |
| Maximum time before ARP table entry becomes stale | Value must be an integer | 150 | The maximum time between an entry in the ARP table being created or refreshed and the entry being removed because it is stale |
| Use string for IP Address | • Disable<br>• Enable | Enable | Take an IP in decimal dot format (for example, "192.168.0.1") as its parameter FreeRTOS_inet_addr_quick() takes an IP address as four separate numerical octets (for example, 192, 168, 0, 1) as its parameters |
| Total number of available network buffers | Value must be an integer | 10 | Define the total number of network buffer that are available to the IP stack |
| Set the maximum number of events | Only symbols, numbers and arithmetic operators are valid. | ipconfigNUM_NETWORK_BUFFER_DESCRIPTORS + 5 | Set the maximum number of events that can be queued for processing at any one time. The event queue must be a minimum of 5 greater than the total number of network buffers |
| Enable FreeRTOS_sendto() without calling Bind | • Enable<br>• Disable | Disable | Set to 1 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the IP stack automatically binding |

| | | | |
|---|---|---|---|
| | | | the socket to a port number from the range socketAUTO_PORT_ALLOCATION_START_NUMBER to 0xffff. If ipconfigALLOW_SOCKET_SEND_WITHOUT_BIND is set to 0 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the send operation being aborted. |
| TTL values for UDP packets | Value must be an integer | 128 | Define the Time To Live (TTL) values used in outgoing UDP packets |
| TTL values for TCP packets | Value must be an integer | 128 | Defines the Time To Live (TTL) values used in outgoing TCP packets |
| Use TCP and all its features | • Disable<br>• Enable | Enable | Use TCP and all its features |
| Let TCP use windowing mechanism | • Disable<br>• Enable | Disable | Let TCP use windowing mechanism |
| Maximum number of bytes the payload of a network frame can contain | Value must be an integer | 1500 | Maximum number of bytes the payload of a network frame can contain |
| Basic DNS client or resolver | • Disable<br>• Enable | Enable | Set ipconfigUSE_DNS to 1 to include a basic DNS client/resolver. DNS is used through the FreeRTOS_gethostbyname() API function. |
| Reply to incoming ICMP echo (ping) requests | • Disable<br>• Enable | Enable | If ipconfigREPLY_TO_INCOMING_PINGS is set to 1 then the IP stack will generate replies to incoming ICMP echo (ping) requests. |
| FreeRTOS_SendPingRequest() is available | • Disable<br>• Enable | Disable | If ipconfigSUPPORT_OUTGOING_PINGS is set to 1 then the FreeRTOS_SendPingRequest() API function is available. |
| FreeRTOS_select() (and associated) API function is available | • Disable<br>• Enable | Disable | If ipconfigSUPPORT_SELECT_FUNCTION is set to 1 then the |

| | | | |
|---|---|---|---|
| | | | FreeRTOS_select() (and associated) API function is available |
| Filter out non Ethernet II frames. | • Disable<br>• Enable | Enable | If ipconfigFILTER_OUT_NON_ETHERNET_II_FRAMES is set to 1 then Ethernet frames that are not in Ethernet II format will be dropped. This option is included for potential future IP stack developments |
| Responsibility of the Ethernet interface to filter out packets | • Disable<br>• Enable | Disable | If ipconfigETHERNET_DRIVER_FILTERS_FRAME_TYPES is set to 1 then it is the responsibility of the Ethernet interface to filter out packets that are of no interest. |
| Access 32-bit fields in the IP packets | Value must be an integer | 2 | To access 32-bit fields in the IP packets with 32-bit memory instructions, all packets will be stored 32-bit-aligned, plus 16-bits. This has to do with the contents of the IP-packets: all 32-bit fields are 32-bit-aligned, plus 16-bit |
| Size of the pool of TCP window descriptors | Value must be an integer | 240 | Define the size of the pool of TCP window descriptors |
| Size of Rx buffer for TCP sockets | Value must be an integer | 3000 | Define the size of Rx buffer for TCP sockets |
| Size of Tx buffer for TCP sockets | Value must be an integer | 3000 | Define the size of Tx buffer for TCP sockets |
| TCP keep-alive | • Disable<br>• Enable | Enable | TCP keep-alive is available or not |
| TCP keep-alive interval | Value must be an integer | 120 | TCP keep-alive interval in second |
| The socket semaphore to unblock the MQTT task (USER_SEMAPHORE) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (WAKE_CALLBACK) | • Disable<br>• Enable | Enable | The socket semaphore is used to unblock the MQTT task |

| | | | |
|---|---|---|---|
| The socket semaphore to unblock the MQTT task (USE_CALLBACKS) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (TX_DRIVER) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| The socket semaphore to unblock the MQTT task (RX_DRIVER) | • Disable<br>• Enable | Disable | The socket semaphore is used to unblock the MQTT task |
| Possible optimisation for expert users | • Disable<br>• Enable | Disable | Possible optimisation for expert users - requires network driver support. It is is useful when there is high network traffic. If non-zero value then instead of passing received packets into the IP task one at a time the network interface can chain received packets together and pass them into the IP task in one go. If set to 0 then only one buffer will be sent at a time. |

**Configurations for Networking > FreeRTOS+TCP Wrapper to ethernet driver (rm_freertos_plus_tcp)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_freertos_plus_tcp0 | Module name. |
| RX Handler Task Stack Size | Only symbols, numbers and arithmetic operators are valid. | configMINIMAL_STACK_SIZE | Stack size of RX handler task. |
| RX Handler Task Priority | Only symbols, numbers and arithmetic operators are valid. | configMAX_PRIORITIES - 1 | Priority of RX handler task. A higher priority ensures timely processing of network traffic or callbacks. |
| Check Link Status Task Stack Size | Only symbols, numbers and arithmetic operators are valid. | configMINIMAL_STACK_SIZE | Stack size of Check link status task. |
| Check Link Status Task Priority | Only symbols, numbers and arithmetic operators are valid. | configMAX_PRIORITIES - 1 | Priority of Check link status task. A low priority is recommended as this is an periodic task. |

| | | | |
|---|---|---|---|
| Link check interval | Only symbols, numbers and arithmetic operators are valid. | 1000 | Interval for checking link status of the ethernet instance of the network interface. Low period will consume more processing time |

# Usage Notes

In order to use the NetworkInterface implementation provided by Renesas for RA devices:

### 1/ In case of Common|Backward Compatible Mode =="NO" (FreeRTOS+TCP 4.0.0 onward)

*Note*

*This mode use IP_Init_Multi to initialize the TCP/IP stack.*
*Before calling this function, at least 1 interface and 1 end-point must have been added to the TCP/IP stack.*

### 2/ In case of Common|Backward Compatible Mode =="YES" (FreeRTOS+TCP 4.0.0 backward)

*Note*

*This mode use IP_Init to initialize stack.*
*Only support 1 interface.*
*Only one instance of the middle port layer should be generated.*

- Some of the stack initialisation APIs have been deprecated as of FreeRTOS V4.0.0 onwards. Please refer to FreeRTOS+TCP Networking Tutorial: Initializing the TCP/IP Stack for the new APIs supporting IPv6, Multiple Endpoints and Multiple interfaces. To use the deprecated APIs please set it to "Yes" for the stack property "Common|Backward Compatible Mode".

*Note*

*The MAC address passed to FreeRTOS_FillEndPoint must match the MAC address configured in the r_ether instance.*
*g_ether_instance must have vEtherISRCallback configured as the callback.*
*The xApplicationGetRandomNumber and ulApplicationGetNextSequenceNumber functions should be implemented in systems using FreeRTOS Plus TCP without Secure Sockets.*
*The weak implementations provided of xApplicationGetRandomNumber and ulApplicationGetNextSequenceNumber rely on rand() as implemented by the respective compiler's math library and may require heap allocation to support re-entrant usage. If these implementations are to be used, srand() must be seeded with a truly random value in the application in order to ensure randomness. This should be done using psa_generate_random() from the mbedtls stack. The weak implementations can also be instead overridden with a user-implementation that uses psa_generate_random().*
*To connect to a server using an IP address the macro ipconfigINCLUDE_FULL_INET_ADDR must be set to 1.*

### Limitations

- Zero-copy is not currently supported by the NetworkInterface.

## 5.2.12.22 GTL BLE Abstraction (rm_ble_abs_gtl)

Modules » Networking

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

# Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE. QE for BLE provides standard and custom services defined by the user.

The module supports the Renesas Electronics DA14531 and DA14535 Bluetooth® Low Energy 5 Modules. The rm_ble_abs_gtl driver interfaces with the DA14531/DA14535 module over UART.

### Features

The Bluetooth® Low Energy Abstraction module with GTL supports the following features:

- Common functionality
  - Boot from host or DA14531/DA14535 flash.
  - Use 1-wire (default) or 2-wire UART for booting.
  - Open/Close the BLE protocol stack.
- The following GAP Role support
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
  - Central: The device that initiates a connection with a peripheral.
  - All roles: The device may act as a central and peripheral the same time.
- GAP functionality
  - Initialize the Host stack.
  - Setting address.
  - Start/Stop Advertising.
  - Connect/Disconnect a link.
  - Scan
- GATT Common functionality
  - Get MTU Size.
- GATT Server functionality
  - Initialization of GATT Server.
  - Loading of Profile definition.
  - Notifications and Indications.
  - Read/Write of GATT Profile from host.
  - Optionally the application and not the device may handle the attributes (Read Indication).
- Gatt Client functionality
  - Discover Peripheral Services/Characteristics.
  - Read/Write Peripheral Characteristics.
  - Receive Notifications and Indications.
- Security functionality (DA14531/DA14535)
  - Supporting Legacy Pairing.
  - Initiate security request procedure from Peripheral as well.
  - Support Secure connections.
  - Support security for both roles (Central - Peripheral).
  - Store bonding data to flash using vee_flash generic module.
  - Enable security from configurator.

    - Store bonding data to flash using vee_flash generic module.
    - Enable security from configurator.
    - Delete all bonding data.
    - Delete bonding data with specific BD address.
    - Read and Write BD address to storage.
    - Add crypto libs to encode bonding data in flash.
    - Add crypto libs to encode bonding data in flash.
- DA1453x Sleep support
    - Three sleep modes are supported with UART HW flow control.

# Configuration

### Pin Configuration

Pins used by the BLE driver to control the DA14531/DA14535 module:

- Reset Pin : Active low reset line for DA14531/DA14535 module.
- UART Interface : RX/TX/RTS Pins. The PMOD DA1453x modules require manually setting CTS GPIO as input. PMOD2_SS should be Input and PMOD2_CLK Output Low.

# Usage Notes

### Getting Started Guide

The below guide walks users through building a fully working solution that allows booting from the host via 1-wire UART in order to run a BLE application from the RA MCU using the GTL interface.

https://lpccs-docs.renesas.com/DA1453x-FSP_BLE_Framework/UM-B-172/index.html

### Addresses

- A public BD address the address pre-programmed into the DA14531/DA14535 and will be used as default.
- A random BD address can be set by calling the R_BLE_VS_SetBdAddr function before the R_BLE_GAP_Init function is called.
- A public BD address can be set by calling the R_BLE_GTL_VS_SetPublicBdAddress() after R_BLE_GAP_Init function is called.

### Heap Requirements

Ensure the BSP heap size is set to at least 2K bytes.

When using FreeRTOS ensure the heap 4 size is set to a minimum of 2K bytes.

### Module Firmware Compatibility

This middleware module is compatible with GTL binary version 6.0.23.1451 and later. You must ensure that the DA14531/DA14535 Module (or PMOD) you are using contains this version (or later) firmware or that you use the boot from host feature and have the host MCU load the binary into the DA14531/DA14535. Note that DA14531 and DA14535 are not firmware compatible even though the GTL API is the same.

Instructions detailing how to upgrade the firmware in a DA14531 Module can be found here:

https://lpccs-docs.renesas.com/US159-DA14531EVZ_Firmware_Upgrade/index.html

The GTL binary file can be downloaded using the tool described in the above instructions, or by using the following link:

https://www.renesas.com/us/en/document/swo/fsp-gtl-binary-us159-da14531evz-pmod-programming?r=1564826

**Limitations**

Developers should be aware of the following limitations when using the BLE_ABS:

Following a power on reset, the R_BLE_VS_GetRand function always returns the same number. Subsequent calls to this function produce random numbers.

Up to 3 concurrent connections are supported.

For DA1453x Sleep mode CTSRTS0 (EK-RA6M4) should be set to P413. Flow Control CTS/RTS Selection should be set to Hardware CTS. If RA is required to wake-up from DA453x P413 and P414 should be shorted and P414 should be set to interrupt mode. R_BLE_GTL_VS_SetSleepMode() is not unsupported, R_BLE_GTL_VS_SetSleepModeExt should be called directly instead.

For bonded devices Client Characteristic Configuration (CCC) values are not saved to flash memory. They are kept to RAM as long as reset does not take place. If the user wishes to save the CCC values to non-volatile memory, he should handle it from the application layer by marking them as Read Indicated (RI, please find the description bellow).

The CCC values are not kept per connection, the value of a CCC is the value of the last modification by a GATT Client. If the user wishes to keep CCC values per connection, he should handle it from the application layer by marking them as Read Indicated (RI, please find the description bellow).

Maximum bonded devices are limited to seven (7). Application must delete an entry before bonding with a new device. Saving bond data is supported with devices that use HP_Flash and boards RA6M4_EK, RA6M5_EK, RA4M3_EK, RA4M2_EK.

Role switching between Central and Peripheral is not supported after the devices have been bonded.

Out Of Band (OOB) paring is not supported.

GAP operations cannot overlap, e.g. you cannot initiate two connections concurrently

The boot from host feature currently supports 1-wire and 2-wire UART operation. If using a 1-wire boot from host the UART RX and TX pins on the host RA MCU must be tied together using a 1K ohm resistor in order to boot the DA14531/DA14535 - this resistor can remain in place after the boot operation has been completed. Boot from host using 2-wire UART is not supported when using a DA14531MOD module because not all the required pins are exposed. Boot from host using 2-wire UART is supported when a DA14535PMOD with SmartBoot (UM-B-171) is already flashed in DA14535. In this case the same pins used for GTL communication are also used for 2-wire boot.

The RI feature of attributes (values are not stored in DA1453x database) is controlled from the flag Enable Characteristic Value Read Callback of QE Tool. In case of a write request the local data base is automatically updated. The app will be notified by an BLE_GATTS_OP_PEER_XXX_WRITE_REQ. In case of a read request and Value Read Callback is enabled (read indication) the local data base is not automatically accessed to get the value. The user is responsible to provide the reply value from gatts_cb by using R_BLE_GTL_GATTS_ReadCfm after a BLE_GATTS_EVENT_DB_ACCESS_IND with

BLE_GATTS_OP_PEER_XXX_READ_REQ. The Client Characteristic Configuration (CCC) values are always treated as RI values. If the attribute is CCC and it is not marked as read_notify, then r_ble_gtl automatically handles it by using a cached value.

Service and characteristic write callback functions, are not generated by the QE Tool. Thus, R_BLE_GTL_GATTC_WRITE_REQ_IND always confirms with R_BLE_GTL_GAP_ERR_NO_ERROR. No value check is provided for incorrect value/range etc

R_BLE_GTL_GAP_StartScan supports only :

- Passive Scan (R_BLE_GTL_GAPM_OP_SCAN_PASSIVE)
- General Discovery Mode (R_BLE_GTL_GAP_GEN_DISCOVERY)
- No White Listing (R_BLE_GTL_SCAN_ALLOW_ADV_ALL)
- Static own address (R_BLE_GTL_GAPM_STATIC_ADDR)
- Slow Scan input
- 1M Phy
- Timeout of 7.7 Seconds restricted by DA145xx ROM.

R_BLE_GTL_GAP_CreateConn supports only :

- Static own address (R_BLE_GTL_GAPM_STATIC_ADDR)
- Automatic connection operation (R_BLE_GTL_GAPM_OP_CONNECTION_AUTO)

Currently supported rm_ble_abs_gtl interface functions:

- RM_BLE_ABS_Open
- RM_BLE_ABS_Close
- RM_BLE_ABS_StartLegacyAdvertising
- RM_BLE_ABS_CreateConnection
- RM_BLE_ABS_StartScanning

Currently supported r_ble_api interface functions:

- R_BLE_Open
- R_BLE_Close
- R_BLE_Execute
- R_BLE_IsTaskFree
- R_BLE_GetVersion
- R_BLE_GAP_Init
- R_BLE_GAP_Terminate
- R_BLE_GAP_UpdConn
- R_BLE_GAP_SetDataLen
- R_BLE_GAP_Disconnect
- R_BLE_GAP_ConfWhiteList
- R_BLE_GAP_GetVerInfo
- R_BLE_GAP_ReadRssi
- R_BLE_GAP_ReadChMap
- R_BLE_GAP_SetAdvParam
- R_BLE_GAP_SetAdvSresData
- R_BLE_GAP_StartAdv
- R_BLE_GAP_StopAdv
- R_BLE_GAP_GetRemainAdvBufSize
- R_BLE_GAP_GetRemDevInfo
- R_BLE_GAP_CreateConn
- R_BLE_GAP_CancelCreateConn

- R_BLE_GAP_StartScan
- R_BLE_GAP_StopScan
- R_BLE_GATTS_SetDbInst
- R_BLE_GATT_GetMtu
- R_BLE_GATTS_RegisterCb
- R_BLE_GATTS_DeregisterCb
- R_BLE_GATTS_Notification
- R_BLE_GATTS_Indication
- R_BLE_GATTS_GetAttr
- R_BLE_GATTS_SetAttr
- R_BLE_GATTC_DiscAllPrimServ
- R_BLE_GATTC_DiscPrimServ
- R_BLE_GATTC_DiscCharByUuid
- R_BLE_GATTC_DiscIncServ
- R_BLE_GATTC_DiscAllChar
- R_BLE_GATTC_DiscAllCharDesc
- R_BLE_GATTC_RegisterCb
- R_BLE_GATTC_DeregisterCb
- R_BLE_GATTC_ReqExMtu
- R_BLE_GATTC_ReadChar
- R_BLE_GATTC_ReadCharUsingUuid
- R_BLE_GATTC_ReadLongChar
- R_BLE_GATTC_ReadMultiChar
- R_BLE_GATTC_WriteCharWithoutRsp
- R_BLE_GATTC_WriteChar
- R_BLE_GATTC_WriteLongChar
- R_BLE_GATTC_ReliableWrites
- R_BLE_GATTC_ExecWrite
- R_BLE_VS_Init
- R_BLE_VS_GetBdAddr
- R_BLE_VS_SetBdAddr
- R_BLE_VS_GetRand
- R_BLE_VS_SetTxPower
- R_BLE_VS_GetTxPower
- R_BLE_GAP_SetPairingParams
- R_BLE_GAP_ReplyPairing
- R_BLE_GAP_ReplyLtkReq
- R_BLE_GAP_ReplyExKeyInfoReq
- R_BLE_GAP_StartPairing
- R_BLE_GAP_ReplyNumComp
- R_BLE_GAP_ReplyPasskeyEntry
- R_BLE_GAP_StartEnc
- R_BLE_GAP_GetDevSecInfo

Currently supported callback events:

- BLE_GAP_EVENT_STACK_ON
- BLE_GAP_EVENT_STACK_OFF
- BLE_GAP_EVENT_CONN_IND
- BLE_GAP_EVENT_DISCONN_IND
- BLE_GAP_EVENT_DATA_LEN_CHG
- BLE_GAP_EVENT_CONN_PARAM_UPD_COMP
- BLE_GAP_EVENT_ADV_ON
- BLE_GAP_EVENT_ADV_OFF
- BLE_GAP_EVENT_ADV_REPT_IND

- BLE_GAP_EVENT_SET_DATA_LEN_COMP
- BLE_GAP_EVENT_LOC_VER_INFO
- BLE_GAP_EVENT_RSSI_RD_COMP
- BLE_GAP_EVENT_CH_MAP_RD_COMP
- BLE_GAP_EVENT_GET_REM_DEV_INFO
- BLE_GAP_EVENT_CONN_CANCEL_COMP
- BLE_GAP_EVENT_CONN_PARAM_UPD_REQ
- BLE_GATTS_EVENT_WRITE_RSP_COMP
- BLE_GATTS_EVENT_HDL_VAL_CNF
- BLE_GATTS_EVENT_DB_ACCESS_IND
- BLE_GATTC_EVENT_EX_MTU_RSP
- BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND
- BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND
- BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP
- BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP
- BLE_GATTC_EVENT_CHAR_READ_RSP
- BLE_GATTC_EVENT_CONN_IND
- BLE_GATTC_EVENT_CHAR_WRITE_RSP
- BLE_GATTC_EVENT_CHAR_WRITE_WITHOUT_RSP
- BLE_GATTC_EVENT_CHAR_DISC_COMP
- BLE_GATTC_EVENT_CHAR_128_DISC_IND
- BLE_GATTC_EVENT_HDL_VAL_IND
- BLE_GATTC_EVENT_HDL_VAL_NTF
- BLE_GATTC_EVENT_EXECUTE_WRITE_COMP
- BLE_GATTC_EVENT_ATT_INFO_REQ_IND
- BLE_GATTC_EVENT_HDL_VAL_SVC_CHNG
- BLE_GAP_EVENT_EX_KEY_REQ
- BLE_GAP_EVENT_PASSKEY_ENTRY_REQ
- BLE_GAP_EVENT_PEER_KEY_INFO
- BLE_GAP_EVENT_PAIRING_COMP
- BLE_GAP_EVENT_LTK_REQ
- BLE_GAP_EVENT_ENC_CHG
- BLE_GAP_EVENT_LTK_RSP_COMP

# Examples

### BLE_ABS_GTL Basic Example

This is a basic example of minimal use of the BLE_ABS_GTL in an application.

```
/* The callback is called when peripheral event occurs. */
void gap_peripheral_cb (uint16_t type, ble_status_t result, st_ble_evt_data_t *
p_data)
{
 FSP_PARAMETER_NOT_USED(result);
 FSP_PARAMETER_NOT_USED(p_data);
 switch (type)
    {
 case BLE_GAP_EVENT_STACK_ON:
```

```
     {
           g_ble_event_flag = g_ble_event_flag | BLE_ABS_EVENT_FLAG_STACK_ON;
 break;
      }
 case BLE_GAP_EVENT_ADV_ON:
     {
 st_ble_gap_adv_set_evt_t * p_gap_adv_set_evt_param = (st_ble_gap_adv_set_evt_t *)
p_data->p_param;
           g_advertising_handle = p_gap_adv_set_evt_param->adv_hdl;
           g_ble_event_flag    |= BLE_ABS_EVENT_FLAG_ADV_ON;
 break;
      }
 case BLE_GAP_EVENT_ADV_OFF:
     {
           g_ble_event_flag |= BLE_ABS_EVENT_FLAG_ADV_OFF;
 break;
      }
 case BLE_GAP_EVENT_CONN_IND:
     {
           g_ble_event_flag |= BLE_ABS_EVENT_FLAG_CONN_IND;
 break;
      }
     {
 /* Do nothing. */
 break;
        }
 default:
 break;
   }
}
```

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 2)
```

```
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 'E', 'x', 'a', 'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 'T', 'E', 'S', 'T', '_', 'E', 'x', 'a',
'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    uint8_t advertising_data[] =
    {
 /* Flags */
       0x02,
       0x01,
       (0x1a),
 /* Shortened Local Name */
       0x08,
       0x08,
       BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
    };
 /* Scan Response Data */
    uint8_t scan_response_data[] =
    {
 /* Complete Local Name */
       0x0D,
       0x09,
       BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
    };
 ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
    {
       .p_peer_address             =
NULL,
```

```
       .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
       .slow_advertising_period    =
0x0000,
       .p_advertising_data         =
advertising_data,
       .advertising_data_length    = sizeof
(advertising_data),
       .p_scan_response_data       =
scan_response_data,
       .scan_response_data_length = sizeof
(scan_response_data),
       .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
       .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
       .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
       .own_bluetooth_address      = {0},
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure the Transmit Level */
    err = (fsp_err_t) R_BLE_VS_SetTxPower(0, BLE_ABS_TRANSMIT_POWER);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_STACK_ON event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
```

```
   time_out_handle_error(timeout);
   g_ble_event_flag = 0;
   timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Start advertising. */
   err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
 while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
   {
 if (BLE_ABS_EVENT_FLAG_ADV_OFF & g_ble_event_flag)
     {
 /* Restart advertise, when stop advertising. */
     err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 if (FSP_SUCCESS == err)
     {
               g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
        }
 else if (FSP_ERR_INVALID_STATE == err)
     {
 /* BLE driver state is busy. */
               ;
        }
 else
     {
 /* Handle any errors. This function should be defined by the user. */
     assert(FSP_SUCCESS == err);
        }
      }
 else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
     {
 /* Stop advertising after a certain amount of time */
 R_BLE_GAP_StopAdv(g_advertising_handle);
```

```
      }
else
     {
          ;
      }
R_BLE_Execute();
   }
   time_out_handle_error(timeout);
/* Clean up & Close BLE driver */
   g_ble_event_flag = 0;
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
}
```

### 5.2.12.23 Layer3 Ethernet Switch (r_layer3_switch)

Modules » Networking

#### Functions

| | |
|---|---|
| fsp_err_t | R_LAYER3_SWITCH_Open (ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_Close (ether_switch_ctrl_t *const p_ctrl)<br>Disables interrupts and stop module. Implements ether_switch_api_t::close. More... |
| fsp_err_t | R_LAYER3_SWITCH_CreateDescriptorQueue (ether_switch_ctrl_t *const p_ctrl, uint32_t *const p_queue_index, const layer3_switch_descriptor_queue_cfg_t *const p_queue_cfg)<br>Create a new descriptor queue and set it to LINKFIX table. This function must be called before calling R_LAYER3_SWITCH_SetDescriptor and R_LAYER3_SWITCH_GetDescriptor. More... |
| fsp_err_t | R_LAYER3_SWITCH_SetDescriptor (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index, layer3_switch_descriptor_t const *const p_descriptor) |

| | |
|---|---|
| | Set descriptor data to a target descriptor. More... |
| fsp_err_t | R_LAYER3_SWITCH_GetDescriptor (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index, layer3_switch_descriptor_t *const p_descriptor)<br>Get descriptor data from a target descriptor. More... |
| fsp_err_t | R_LAYER3_SWITCH_StartDescriptorQueue (ether_switch_ctrl_t *const p_ctrl, uint32_t queue_index)<br>Reload and enable a descriptor queue. In a TX descriptor queue, the queue start transmission. In a RX descriptor queue, the queue start reception. More... |
| fsp_err_t | R_LAYER3_SWITCH_CallbackSet (ether_switch_ctrl_t *const p_ctrl, void(*p_callback)(ether_switch_callback_args_t *), void *const p_context, ether_switch_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_LAYER3_SWITCH_ConfigurePort (ether_switch_ctrl_t *const p_ctrl, uint8_t port, layer3_switch_port_cfg_t *p_port_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_AddTableEntry (ether_switch_ctrl_t *const p_ctrl, layer3_switch_frame_filter_t const *const p_target_frame, layer3_switch_table_entry_cfg_t const *const p_entry_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_SearchTableEntry (ether_switch_ctrl_t *const p_ctrl, layer3_switch_frame_filter_t const *const p_target_frame, layer3_switch_table_entry_cfg_t *const p_entry_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_ConfigureTable (ether_switch_ctrl_t *const p_ctrl, layer3_switch_table_cfg_t const *const p_table_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_GetTable (ether_switch_ctrl_t *const p_ctrl, layer3_switch_table_t *const p_table) |
| fsp_err_t | R_LAYER3_SWITCH_ConfigureTAS (ether_switch_ctrl_t *const p_ctrl, uint8_t port, layer3_switch_tas_cfg_t *p_tas_cfg) |
| fsp_err_t | R_LAYER3_SWITCH_EnableTAS (ether_switch_ctrl_t *const p_ctrl, uint8_t port) |

### Detailed Description

The LAYER3 SWITCH module (r_layer3_switch) provides an API for standard Ethernet communications applications that use the ESWM peripheral. It implements the Ethernet Switch Interface.

# Overview

This module performs Ethernet frame transmission and reception using Ethernet Switch Module.

The following figure shows which IPs are used by the r_layer3_switch module.

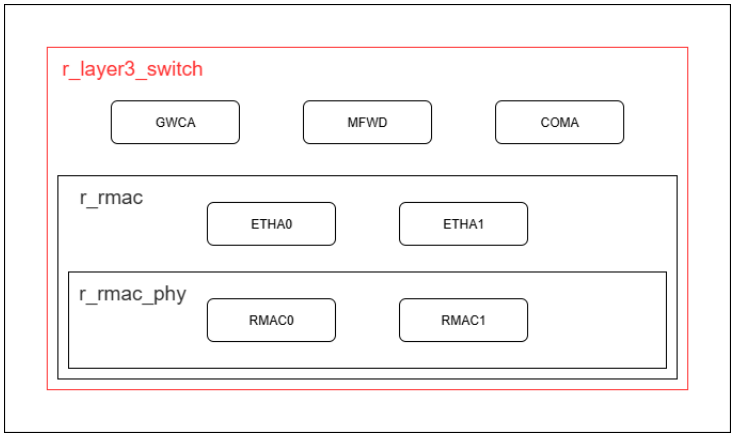



Figure 323: ESWM block diagram

### Features

The layer3 switch module supports the following features:

- Transmit/receive data using a descriptor.
- Timestamp feature
- Optional callback function when transmit/receive complete or an error occurs
- Frame Replication and Elimination for Reliability (FRER) feature.

# Configuration

### Build Time Configurations for r_layer3_switch

The following build time configurations are defined in fsp_cfg/r_layer3_switch_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Available queue num | Value must be an integer between 1 and 64 | 4 | available queue num. |
| gPTP enable | • Enabled<br>• Disabled | Enabled | Select whether or not to use gPTP function. |
| Time Aware Shaper | • Enabled<br>• Disabled | Disabled | Select whether or not to use the Time Aware Shaper function. |

### Configurations for Networking > Switch (r_layer3_switch)

This module can be added to the Stacks tab via New Stack > Networking > Switch (r_layer3_switch).

Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Interrupts | | | |
| Interrupt priority | MCU Specific Options | | Select the layer3_switch interrupt priority. |
| ETHA0 error interrupt priority | MCU Specific Options | | Select the layer3_switch etha0 interrupt priority. |
| ETHA1 error interrupt priority | MCU Specific Options | | Select the layer3_switch etha1 interrupt priority. |
| Ports | | | |
| Ports > Port 0 | | | |
| Ports > Port 0 > IPV queue depth | | | |
| Queue 0 | Value must be an integer more than 1 | 64 | queue0 dept of port 0. |
| Queue 1 | Value must be an integer more than 1 | 64 | queue1 dept of port 0. |
| Queue 2 | Value must be an integer more than 1 | 64 | queue2 dept of port 0. |
| Queue 3 | Value must be an integer more than 1 | 64 | queue3 dept of port 0. |
| Queue 4 | Value must be an integer more than 1 | 64 | queue4 dept of port 0. |
| Queue 5 | Value must be an integer more than 1 | 64 | queue5 dept of port 0. |
| Queue 6 | Value must be an integer more than 1 | 64 | queue6 dept of port 0. |
| Queue 7 | Value must be an integer more than 1 | 64 | queue7 dept of port 0. |
| Fowarding | MCU Specific Options | | Destination ports for port 0 fowarding |
| MAC address | Must be a valid MAC address | 00:11:22:33:44:55 | MAC address of port 0. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs for port0. |
| Context | Name must be a valid C symbol | NULL | Placeholder for user data. Passed to the user callback in |

| | | | |
|---|---|---|---|
| | | | ether_switch_callback_args_t for port0. |
| GPTP timer number | • Timer 0<br>• Timer 1 | Timer 0 | GPTP timer number of port 0. |
| Ports > Port 1 | | | |
| Ports > Port 1 > IPV queue depth | | | |
| Queue 0 | Value must be an integer more than 1 | 64 | queue0 dept of port 1. |
| Queue 1 | Value must be an integer more than 1 | 64 | queue1 dept of port 1. |
| Queue 2 | Value must be an integer more than 1 | 64 | queue2 dept of port 1. |
| Queue 3 | Value must be an integer more than 1 | 64 | queue3 dept of port 1. |
| Queue 4 | Value must be an integer more than 1 | 64 | queue4 dept of port 1. |
| Queue 5 | Value must be an integer more than 1 | 64 | queue5 dept of port 1. |
| Queue 6 | Value must be an integer more than 1 | 64 | queue6 dept of port 1. |
| Queue 7 | Value must be an integer more than 1 | 64 | queue7 dept of port 1. |
| Fowarding | MCU Specific Options | | Destination ports for port 1 fowarding |
| MAC address | Must be a valid MAC address | 00:11:22:33:44:55 | MAC address of port 1. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs for port1. |
| Context | Name must be a valid C symbol | NULL | Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t for port1. |
| GPTP timer number | • Timer 0<br>• Timer 1 | Timer 1 | GPTP timer number of port 1. |
| CBS | | | |
| CBS > Port 0 Band width ( % ) | | | |
| Queue 0 | Value must be an integer between 0 and 100 | 0 | queue0 Band width of port 0. |
| Queue 1 | Value must be an | 0 | queue1 Band width of |

| | | | |
|---|---|---|---|
| | integer between 0 and 100 | | port 0. |
| Queue 2 | Value must be an integer between 0 and 100 | 0 | queue2 Band width of port 0. |
| Queue 3 | Value must be an integer between 0 and 100 | 0 | queue3 Band width of port 0. |
| Queue 4 | Value must be an integer between 0 and 100 | 0 | queue4 Band width of port 0. |
| Queue 5 | Value must be an integer between 0 and 100 | 0 | queue5 Band width of port 0. |
| Queue 6 | Value must be an integer between 0 and 100 | 0 | queue6 Band width of port 0. |
| Queue 7 | Value must be an integer between 0 and 100 | 0 | queue7 Band width of port 0. |
| CBS > Port 1 Band width ( % ) | | | |
| Queue 0 | Value must be an integer between 0 and 100 | 0 | queue0 Band width of port 1. |
| Queue 1 | Value must be an integer between 0 and 100 | 0 | queue1 Band width of port 1. |
| Queue 2 | Value must be an integer between 0 and 100 | 0 | queue2 Band width of port 1. |
| Queue 3 | Value must be an integer between 0 and 100 | 0 | queue3 Band width of port 1. |
| Queue 4 | Value must be an integer between 0 and 100 | 0 | queue4 Band width of port 1. |
| Queue 5 | Value must be an integer between 0 and 100 | 0 | queue5 Band width of port 1. |
| Queue 6 | Value must be an integer between 0 and 100 | 0 | queue6 Band width of port 1. |
| Queue 7 | Value must be an integer between 0 and 100 | 0 | queue7 Band width of port 1. |

CBS > Port 0 Maximum burst frame number

| | | | |
|---|---|---|---|
| Queue 0 | Value must be an integer between 0 and 255 | 0 | queue0 maximum burst frame number of port 0. |
| Queue 1 | Value must be an integer between 0 and 255 | 0 | queue1 maximum burst frame number of port 0. |
| Queue 2 | Value must be an integer between 0 and 255 | 0 | queue2 maximum burst frame number of port 0. |
| Queue 3 | Value must be an integer between 0 and 255 | 0 | queue3 maximum burst frame number of port 0. |
| Queue 4 | Value must be an integer between 0 and 255 | 0 | queue4 maximum burst frame number of port 0. |
| Queue 5 | Value must be an integer between 0 and 255 | 0 | queue5 maximum burst frame number of port 0. |
| Queue 6 | Value must be an integer between 0 and 255 | 0 | queue6 maximum burst frame number of port 0. |
| Queue 7 | Value must be an integer between 0 and 255 | 0 | queue7 maximum burst frame number of port 0. |

CBS > Port 1 Maximum burst frame number

| | | | |
|---|---|---|---|
| Queue 0 | Value must be an integer between 0 and 255 | 0 | queue0 maximum burst frame number of port 0. |
| Queue 1 | Value must be an integer between 0 and 255 | 0 | queue1 maximum burst frame number of port 0. |
| Queue 2 | Value must be an integer between 0 and 255 | 0 | queue2 maximum burst frame number of port 0. |
| Queue 3 | Value must be an integer between 0 and 255 | 0 | queue3 maximum burst frame number of port 0. |
| Queue 4 | Value must be an integer between 0 and 255 | 0 | queue4 maximum burst frame number of port 0. |
| Queue 5 | Value must be an integer between 0 and 255 | 0 | queue5 maximum burst frame number of port 0. |
| Queue 6 | Value must be an | 0 | queue6 maximum |

| | | | |
|---|---|---|---|
| | integer between 0 and 255 | | burst frame number of port 0. |
| Queue 7 | Value must be an integer between 0 and 255 | 0 | queue7 maximum burst frame number of port 0. |
| Name | Name must be a valid C symbol | g_layer3_switch0 | Module name. |
| Channel | 0 | 0 | Select the layer3 switch controller channel number. |
| Context | Name must be a valid C symbol | NULL | Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t. |
| Callback | Name must be a valid C symbol | NULL | Callback provided when an ISR occurs |
| Number of Layer 3 routing entries | Value must be an integer between 1 and 256 | 10 | Maximum number of Layer 3 routing entries. |

# Usage Notes

### Time-Sensitive Network (TSN)

### Credit Based Shaper [802.1Qav]

Credit Based Shaper (CBS) is a standard defined in IEEE 802.1Qav that limits the throughput of Ethernet traffic. The figure below illustrates the operation of CBS when the bandwidth is set to 40Mbps. When the transmission throughput exceeds the bandwidth configured by CBS, frame transmission is restricted. The restricted frames are buffered in the queue and transmitted with delay.

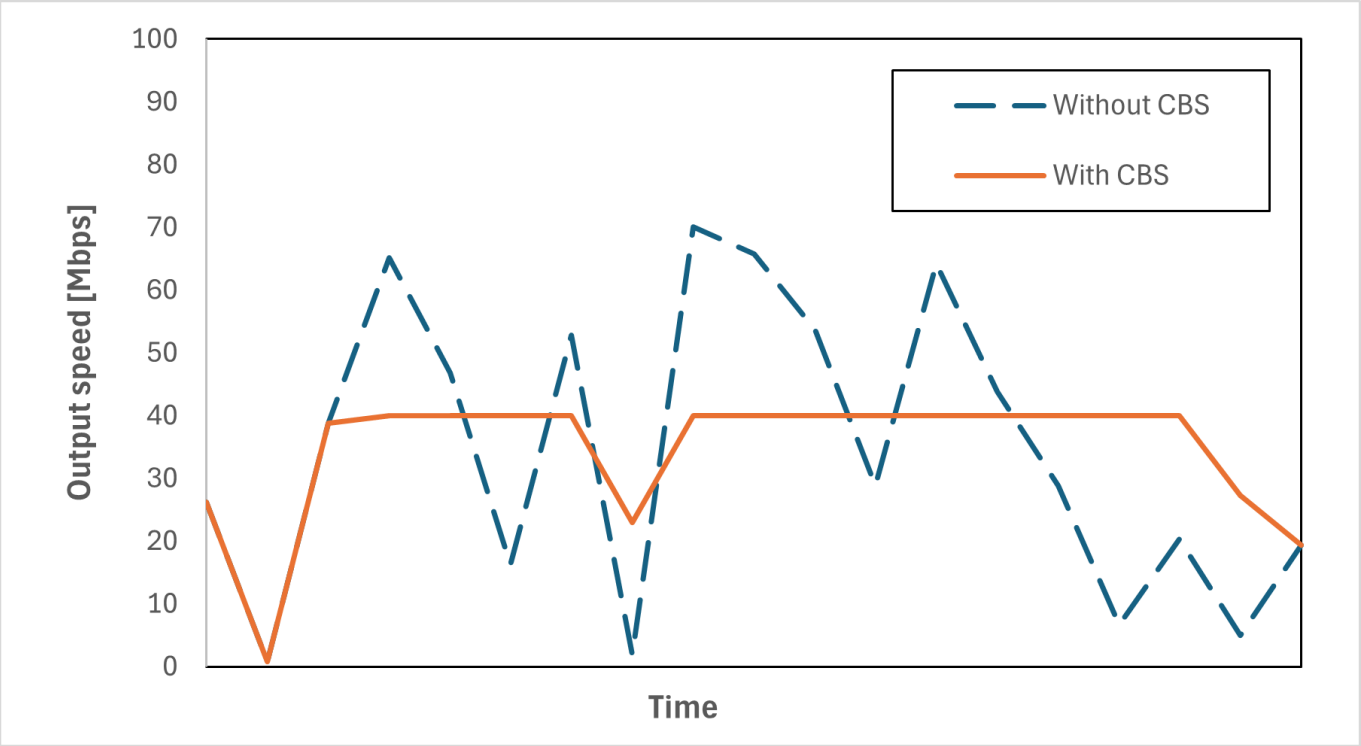



Figure 324: CBS operation image

In the r_layer3_switch module, CBS bandwidth can be configured for each Internal Priority Value (IPV) queue per transmission port. There are 8 IPV queues available for each port. On transmission ports where CBS is enabled, the throughput is limited to the configured bandwidth. CBS can be configured through stack properties or the ConfigurePort() API.

*Note*

*The maximum number of frames that can be buffered by CBS is limited to the depth of each IPV queue. If many frames are buffered in high-priority queues, the common memory pool may be depleted, potentially limiting transmission from other queues.*

### Time-Aware Shaper [802.1Qbv]

Time-Aware Shaper (TAS) is a standard defined in IEEE 802.1Qbv that limits Ethernet traffic by controlling the time window when frames are transmitted. TAS controls transmission by changing gate states. Frames are transmitted while the gate is opened and when the gate is closed, frames cannot be transmitted and are delayed until the next open state. By opening each gate at different timings, it is possible to prevent frames of specific priorities from monopolizing the bandwidth. The figure below illustrates the operation of TAS. In this image, the gate cycle time is set to 10ms, Gate 0 is opened for 2.5 ms and closed for 7.5 ms, Gate 1 is closed for 1.0 + 5.0 ms and open for 4.0 ms.

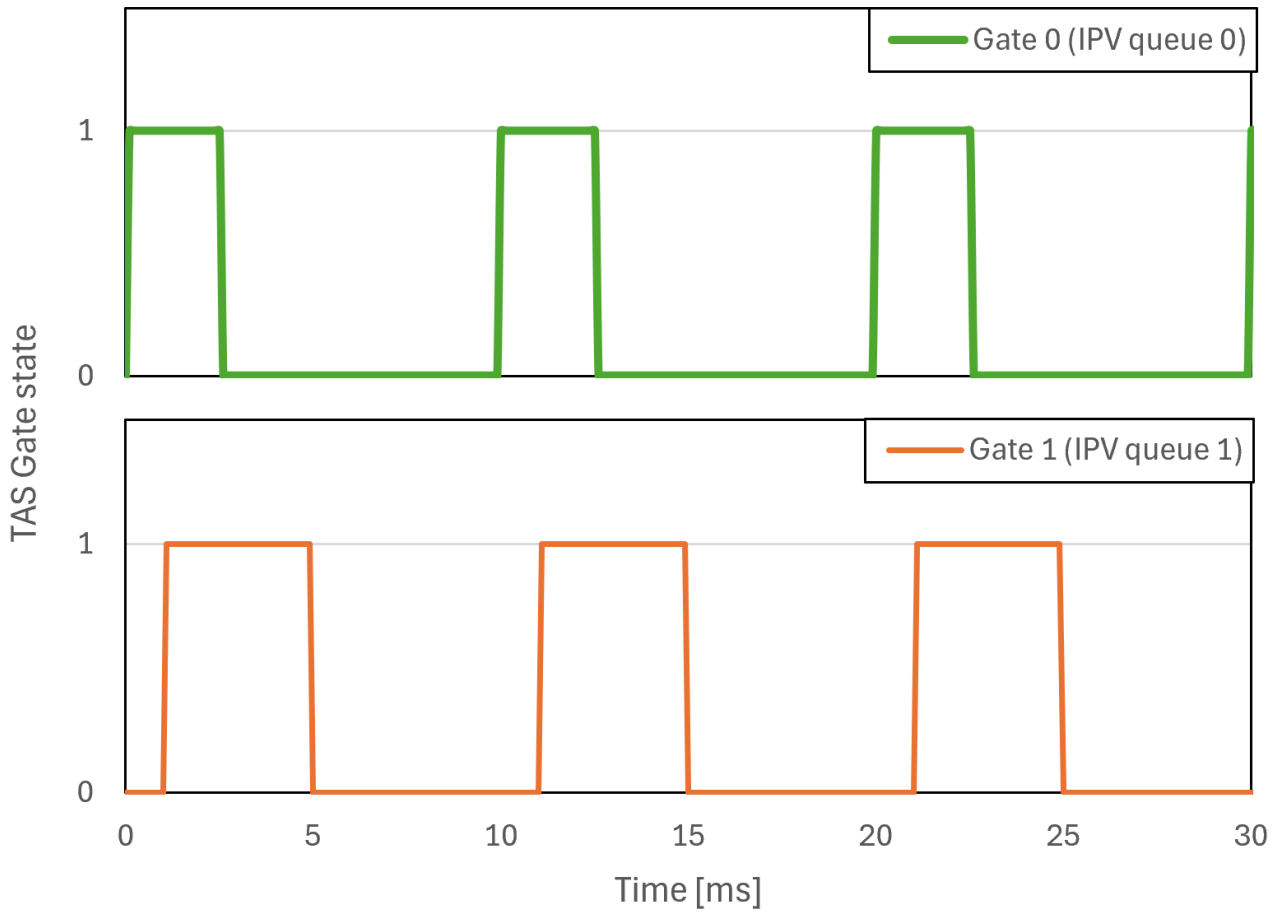



Figure 325: TAS operation image

In the r_layer3_switch module, TAS gates can be configured for each IPV queue of the transmission ports. Each gate learns entries that represent windows. An entry contains the window time and state. TAS feature is configured using the ConfigureTAS() API, and enabled using the EnableTAS() API.

*Note*

*This TAS feature uses the gPTP timer.*

### Limitations

Developers should be aware of the following limitations when using the r_layer3_switch:

- CBS feature supports only 100 Mbps and 1 Gbps link speeds.

# Examples

### Basic Example

This is a basic example of transmission use of the LAYER3 SWITCH.

```
#define LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE (8)
```

```
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE (1514)
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER ((uint8_t)
((LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE & \
  0xFF00) >> 8))
#define LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER ((uint8_t)
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE & \
  0x00FF)
#define LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define LAYER3_SWITCH_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define LAYER3_SWITCH_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79,
0x02
#define LAYER3_SWITCH_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define LAYER3_SWITCH_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
layer3_switch_descriptor_t
gp_descriptor_pool[LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE];
uint8_t gp_ether_buffer0[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer1[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer2[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer3[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer4[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer5[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer6[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t gp_ether_buffer7[LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE];
uint8_t * gpp_ether_buffers[8] =
{
    &gp_ether_buffer0[0],
    &gp_ether_buffer1[0],
    &gp_ether_buffer2[0],
    &gp_ether_buffer3[0],
```

```
    &gp_ether_buffer4[0],
    &gp_ether_buffer5[0],
    &gp_ether_buffer6[0],
    &gp_ether_buffer7[0],
};
/* Transmit data buffer */
uint8_t gp_send_data[LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    LAYER3_SWITCH_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    LAYER3_SWITCH_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    LAYER3_SWITCH_EXAMPLE_FRAME_TYPE,             /* Type field */
    LAYER3_SWITCH_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void layer3_switch_transmit_example (void)
{
 fsp_err_t err         = FSP_SUCCESS;
    uint32_t queue_index = 0;
 /* Configure transmission queue. */
 layer3_switch_descriptor_queue_cfg_t transmission_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_TX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_descriptor_pool,
       .p_ts_descriptor_array = NULL,
       .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
       .array_length          = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
       .ports                = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Prepare data for transmission. */
    memcpy(gpp_ether_buffers[0], gp_send_data,
LAYER3_SWITCH_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE);
 /* Initialize transmission descriptor. */
 layer3_switch_descriptor_t transmission_descriptor = {0};
```

```
    transmission_descriptor.basic.ds_l   =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
    transmission_descriptor.basic.ds_h   =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    transmission_descriptor.basic.die    = 1;
    transmission_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
    transmission_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE;
    transmission_descriptor.info1_tx.fmt = 1;
    transmission_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize PHY-LSI chip. */
    err = R_RMAC_PHY_ChipInit(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Get the Ethernet link status read from the PHY-LSI Basic Status registers until
link-up. */
       err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    } while (FSP_SUCCESS != err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&transmission_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Set descriptor to queue. */
    err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
```

```
 /* Start transmission. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
 /* Get descriptor status. */
    err = R_LAYER3_SWITCH_GetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&transmission_descriptor);
    assert(FSP_SUCCESS == err);
 /* Check if transmission is completed. */
    assert(LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY == transmission_descriptor.basic.dt);
 /* Disable transmission and receive function and close the layer3 switch instance.
*/
 R_LAYER3_SWITCH_Close(&g_layer3_switch0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

This is a basic example of reception use of the LAYER3 SWITCH.

```
#define LAYER3_SWITCH_EXAMPLE_RECEPTION_QUEUE_LENGTH (4)
bool g_complete_reception_flag;
void layer3_switch_receive_example (void)
{
 fsp_err_t err         = FSP_SUCCESS;
    uint32_t queue_index = 1;
 /* Configure reception queue. */
 layer3_switch_descriptor_queue_cfg_t reception_queue_cfg =
    {
       .descriptor_format    = LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED,
       .type                  = LAYER3_SWITCH_QUEUE_TYPE_RX,
       .write_back_mode       = LAYER3_SWITCH_WRITE_BACK_MODE_FULL,
       .p_descriptor_array    = gp_descriptor_pool,
       .p_ts_descriptor_array = NULL,
       .rx_timestamp_storage = LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE,
       .array_length          = LAYER3_SWITCH_EXAMPLE_DESCRIPTOR_POOL_SIZE,
```

```
       .ports                = LAYER3_SWITCH_PORT_BITMASK_PORT0,
    };
 /* Initialize reception descriptor. */
 layer3_switch_descriptor_t reception_descriptor = {0};
    reception_descriptor.basic.ds_l   =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_LOWER;
    reception_descriptor.basic.ds_h   =
LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE_UPPER;
    reception_descriptor.basic.die    = 1;
    reception_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers;
    reception_descriptor.basic.dt    = LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY;
    reception_descriptor.info1_tx.fmt = 1;
    reception_descriptor.info1_tx.dv = LAYER3_SWITCH_PORT_BITMASK_PORT0;
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Initialize PHY-LSI chip. */
    err = R_RMAC_PHY_ChipInit(&g_rmac_phy0_ctrl, &g_rmac_phy0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start auto negotiation. */
    err = R_RMAC_PHY_StartAutoNegotiate(&g_rmac_phy0_ctrl);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Get the Ethernet link status read from the PHY-LSI Basic Status registers until
link-up. */
       err = R_RMAC_PHY_LinkStatusGet(&g_rmac_phy0_ctrl);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    } while (FSP_SUCCESS != err);
 /* Create descriptor queue. */
    err = R_LAYER3_SWITCH_CreateDescriptorQueue(&g_layer3_switch0_ctrl, &queue_index,
&reception_queue_cfg);
    assert(FSP_SUCCESS == err);
 /* Copy descriptor to queue. */
```

```
 for (uint16_t i = 0; i < LAYER3_SWITCH_EXAMPLE_RECEPTION_QUEUE_LENGTH - 1; i++)
    {
       memset(gpp_ether_buffers[i], 0, LAYER3_SWITCH_EXAMPLE_MAXIMUM_FRAME_SIZE);
       reception_descriptor.basic.ptr_l = (uintptr_t) gpp_ether_buffers[i];
       err = R_LAYER3_SWITCH_SetDescriptor(&g_layer3_switch0_ctrl, queue_index,
&reception_descriptor);
      assert(FSP_SUCCESS == err);
    }
    g_complete_reception_flag = false;
 /* Start reception. */
    err = R_LAYER3_SWITCH_StartDescriptorQueue(&g_layer3_switch0_ctrl, queue_index);
    assert(FSP_SUCCESS == err);
 /* Wait to complete reception. */
 while (true != g_complete_reception_flag)
    {
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    }
 /* Disable transmission and receive function and close the layer3 switch instance.
*/
 R_LAYER3_SWITCH_Close(&g_layer3_switch0_ctrl);
    assert(FSP_SUCCESS == err);
}
void layer3_switch_example_callback (ether_switch_callback_args_t * p_args) {
 switch (p_args->event)
    {
 case ETHER_SWITCH_EVENT_RX_COMPLETE:
     {
           g_complete_reception_flag = true;
 break;
     }
 case ETHER_SWITCH_EVENT_TX_COMPLETE:
     {
 break;
     }
```

```
 case ETHER_SWITCH_EVENT_RX_QUEUE_FULL:
      {
 break;
      }
 case ETHER_SWITCH_EVENT_RX_MESSAGE_LOST:
      {
 break;
      }
 default:
      {
 break;
      }
    }
}
```

### TSN Example

The below is an example of forwarding frames with Credit Based Shaper (CBS).

```
void layer3_switch_cbs_example (void)
{
 fsp_err_t                err             = FSP_SUCCESS;
    uint8_t                  source_port      = 0;
    uint8_t                  destination_port = 1;
 layer3_switch_port_cfg_t port_cfg         = {0};
 /* Set bandwidth of queue 0 to 30 Mbps, queue 1 to 20 Mbps. */
 layer3_switch_cbs_cfg_t cbs_cfg =
    {
       .band_width_list    = {30, 20, 0, 0, 0, 0, 0, 0   },
       .max_burst_num_list = {10, 10, 10, 10, 10, 10, 10, 10 },
    };
 /* Destination of forwarding target frames are 00:01:02:03:04:05 and
10:11:12:13:14:15*/
    uint8_t p_mac_address_list[2][6] =
    {
```

```
      {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, },
      {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, },
   };
 layer3_switch_table_cfg_t mac_table_cfg =
   {
      .port_cfg_list             = {{.mac_table_enable = true}, {.mac_table_enable =
true}},
      .unsecure_entry_maximum_num = 2,
   };
 layer3_switch_frame_filter_t    mac_entry_frame;
 layer3_switch_table_entry_cfg_t mac_entry_cfg =
   {
      .destination_ports              = (1 << destination_port),
      .source_ports                  = (1 << source_port),
      .internal_priority_update_enable = true,
      .entry_enable                  = true,
   };
 /* Open the layer3 switch instance with initial configuration. */
   err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
   assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
   mac_entry_frame.entry_type = LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC;
 /* Reset MAC table. */
   err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl, &mac_table_cfg);
   assert(FSP_SUCCESS == err);
 /* Learn destination MAC address of a forwarding frame. */
 for (uint8_t i = 0; i < 2; i++)
   {
      mac_entry_frame.p_destination_mac_address    = p_mac_address_list[i];
      mac_entry_cfg.internal_priority_update_value = i;
      err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl, &mac_entry_frame,
&mac_entry_cfg);
     assert(FSP_SUCCESS == err);
   }
```

```
 /* Configure and enable CBS. */
    port_cfg.p_cbs_cfg = &cbs_cfg;
    err                = R_LAYER3_SWITCH_ConfigurePort(&g_layer3_switch0_ctrl,
destination_port, &port_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

The below is an example of forwarding frames with Credit Based Shaper (TAS).

```
#define LAYER3_SWITCH_EXAMPLE_GPTP_TIMER (0)
#define LAYER3_SWITCH_EXAMPLE_GPTP_CLOCK_PERIOD (4)
void layer3_switch_tas_example (void)
{
 fsp_err_t err             = FSP_SUCCESS;
    uint8_t   source_port      = 0;
    uint8_t   destination_port = 1;
 layer3_switch_table_entry_t mac_table_entry;
 layer3_switch_tas_cfg_t      tas_cfg;
 layer3_switch_tas_entry_t    closed_entry_list[1] = {{.state =
LAYER3_SWITCH_TAS_GATE_STATE_CLOSE, .time = 0}};
 layer3_switch_tas_gate_cfg_t tas_gate_closed      =
    {
       .tas_entry_num    = 1, .initial_gate_state =
LAYER3_SWITCH_TAS_GATE_STATE_CLOSE,
       .p_tas_entry_list = closed_entry_list
    };
 gptp_instance_t const * p_gptp_instance =
g_layer3_switch0_extended_cfg.p_gptp_instance;
 gptp_timer_cfg_t       timer_cfg       = {.clock_period =
LAYER3_SWITCH_EXAMPLE_GPTP_CLOCK_PERIOD};
```

```
 /* Open the layer3 switch instance with initial configuration. */
    err = R_LAYER3_SWITCH_Open(&g_layer3_switch0_ctrl, &g_layer3_switch0_cfg);
    assert(FSP_SUCCESS == err);
 /* Gate 0 and 1 are repeat Open/Close. Other gates are alwayes closed. */
    tas_cfg.gate_cfg_list[0] = g_layer3_switch_example_tas_gate_0_cfg;
    tas_cfg.gate_cfg_list[1] = g_layer3_switch_example_tas_gate_1_cfg;
 for (uint8_t i = 2; i < 8; i++)
    {
       tas_cfg.gate_cfg_list[i] = tas_gate_closed;
    }
    tas_cfg.gptp_timer_number    = LAYER3_SWITCH_EXAMPLE_GPTP_TIMER;
    tas_cfg.cycle_time_start_high =
LAYER3_SWITCH_EXAMPLE_CYCLE_START_TIME_UPPER_32BIT;
    tas_cfg.cycle_time_start_low =
LAYER3_SWITCH_EXAMPLE_CYCLE_START_TIME_UPPER_32BIT;
    tas_cfg.cycle_time            = LAYER3_SWITCH_EXAMPLE_CYCLE_TIME;
 /* Start gPTP timer before enable TAS feature. */
    p_gptp_instance->p_api->timerCfg(p_gptp_instance->p_ctrl,
LAYER3_SWITCH_EXAMPLE_GPTP_TIMER, &timer_cfg);
    p_gptp_instance->p_api->start(p_gptp_instance->p_ctrl,
LAYER3_SWITCH_EXAMPLE_GPTP_TIMER);
 /* Configure and enable TAS. */
    err = R_LAYER3_SWITCH_ConfigureTAS(&g_layer3_switch0_ctrl, destination_port,
&tas_cfg);
    assert(FSP_SUCCESS == err);
    err = R_LAYER3_SWITCH_EnableTAS(&g_layer3_switch0_ctrl, destination_port);
    assert(FSP_SUCCESS == err);
 /* Initialize MAC table entry.*/
    err = R_LAYER3_SWITCH_ConfigureTable(&g_layer3_switch0_ctrl,
&g_layer3_switch_example_mac_table_cfg);
    assert(FSP_SUCCESS == err);
 /* Learn a MAC entry. */
    mac_table_entry.target_frame.entry_type                  =
LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC;
```

```
    mac_table_entry.entry_cfg.entry_enable                    = true;
    mac_table_entry.entry_cfg.security_enable                 = true;
    mac_table_entry.entry_cfg.destination_ports                = (uint32_t) (1 <<
destination_port);
    mac_table_entry.entry_cfg.source_ports                    = (1 << source_port);
    mac_table_entry.entry_cfg.internal_priority_update_enable = true;
 for (uint8_t i = 0; i < 2; i++)
    {
       mac_table_entry.target_frame.p_destination_mac_address   =
g_layer3_switch_example_mac_address_list[i];
       mac_table_entry.entry_cfg.internal_priority_update_value = i;
       err = R_LAYER3_SWITCH_AddTableEntry(&g_layer3_switch0_ctrl,
                                           &mac_table_entry.target_frame,
                                           &mac_table_entry.entry_cfg);
      assert(FSP_SUCCESS == err);
    }
 while (1)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | layer3_switch_basic_descriptor_t |
| struct | layer3_switch_descriptor_t |
| struct | layer3_switch_ts_reception_process_descriptor_t |
| struct | layer3_switch_frer_cfg_t |
| struct | layer3_switch_frer_entry_t |
| struct | layer3_switch_frer_entry_cfg_t |
| struct | layer3_switch_frer_sequence_recovery_status_t |
| struct | layer3_switch_descriptor_queue_cfg_t |

| | |
|---|---|
| struct | layer3_switch_cbs_cfg_t |
| struct | layer3_switch_port_cfg_t |
| struct | layer3_switch_descriptor_queue_status_t |
| struct | layer3_switch_ipv6_filter_address_offset_t |
| struct | layer3_switch_l3_stream_filter_cfg_t |
| struct | layer3_switch_frame_vlan_tag_t |
| struct | layer3_switch_frame_filter_t |
| struct | layer3_switch_l3_filter_t |
| struct | layer3_switch_l3_update_config_t |
| struct | layer3_switch_table_entry_cfg_t |
| struct | layer3_switch_table_entry_t |
| struct | layer3_switch_forwarding_port_cfg_t |
| struct | layer3_switch_table_t |
| struct | layer3_switch_table_cfg_t |
| struct | layer3_switch_extended_cfg_t |
| struct | layer3_switch_instance_ctrl_t |
| struct | layer3_switch_tas_entry_t |
| struct | layer3_switch_tas_gate_cfg_t |
| struct | layer3_switch_tas_cfg_t |

## Enumerations

| | |
|---|---|
| enum | layer3_switch_descriptor_type_t |
| enum | layer3_switch_port_bitmask_t |
| enum | layer3_switch_queue_type_t |
| enum | layer3_switch_descriptor_format_t |
| enum | layer3_switch_write_back_mode_t |

| | |
|---|---|
| enum | layer3_switch_ip_version_t |
| enum | layer3_switch_ip_address_direction_t |
| enum | layer3_switch_magic_packet_detection_t |
| enum | layer3_switch_vlan_mode_t |
| enum | layer3_switch_vlan_ingress_mode_t |
| enum | layer3_switch_vlan_egress_mode_t |
| enum | layer3_switch_ip_protocol_t |
| enum | layer3_switch_forwarding_r_tag_t |
| enum | layer3_switch_l3_filter_bitmask_t |
| enum | layer3_switch_l3_update_bitmask_t |
| enum | layer3_switch_table_entry_type_t |
| enum | layer3_switch_table_status_t |
| enum | layer3_switch_tas_gate_state_t |
| enum | layer3_switch_rx_timestamp_storage_t |
| enum | layer3_switch_ts_descriptor_queue_status_t |

## Data Structure Documentation

### ◆ layer3_switch_basic_descriptor_t

| struct layer3_switch_basic_descriptor_t |
|---|
| GWCA descriptor. |

### ◆ layer3_switch_descriptor_t

| struct layer3_switch_descriptor_t |
|---|
| GWCA extended descriptor. |

### ◆ layer3_switch_ts_reception_process_descriptor_t

| struct layer3_switch_ts_reception_process_descriptor_t |
|---|
| GWCA TS reception process descriptor. |

### ◆ layer3_switch_frer_cfg_t

| struct layer3_switch_frer_cfg_t | | |
|---|---|---|
| Configuration of FRER feature. | | |
| Data Fields | | |
| uint32_t | sys_clock | Timeout check time. |
| uint32_t | timeout_enable | Timeout check valid. |
| uint32_t | check_period | Microsecond prescaler used to creates an internal clock for aging at 1 MHz to derive the timeout 1 kHz clock. |

### ◆ layer3_switch_frer_entry_t

| struct layer3_switch_frer_entry_t | | |
|---|---|---|
| FRER table entry. | | |
| Data Fields | | |
| uint32_t | take_no_sequence | Reject frames without R-TAG. |
| uint32_t | sequence_history_len | Sequence history length. |
| uint32_t | set_recovery_remaining_tick | Time in tick before timeout. |

### ◆ layer3_switch_frer_entry_cfg_t

| struct layer3_switch_frer_entry_cfg_t | | |
|---|---|---|
| Configuration of FRER entry (individual and sequence recovery). | | |
| Data Fields | | |
| layer3_switch_frer_entry_t | individual_recovery | FRER table entry for individual recovery. |
| layer3_switch_frer_entry_t * | p_sequence_recovery | Pointer to FRER entry(sequence recovery) information referenced by FRERN in this FRER entry(individual recovery). |
| uint32_t | sequence_recovery_id | |

### ◆ layer3_switch_frer_sequence_recovery_status_t

| struct layer3_switch_frer_sequence_recovery_status_t | | |
|---|---|---|
| Status of FRER sequence recovery table. | | |
| Data Fields | | |
| bool | learned | Flags whether the FRER entry(sequence recovery) related to this entry has been |

| | | |
|---|---|---|
| | | learned. |
| uint32_t | frer_entry_index | Real FRER table index of the FRER entry(sequence recovery) related to this entry. |

### ◆ layer3_switch_descriptor_queue_cfg_t

| struct layer3_switch_descriptor_queue_cfg_t | | |
|---|---|---|
| Configuration of a descriptor queue. | | |
| Data Fields | | |
| layer3_switch_descriptor_format_t | descriptor_format | Enable or disable extended descriptors. |
| layer3_switch_queue_type_t | type | Reception queue or transmission queue. |
| layer3_switch_write_back_mode_t | write_back_mode | Configure write-back to descriptor fields. |
| layer3_switch_descriptor_t * | p_descriptor_array | Descriptor array that use to create queue. |
| layer3_switch_ts_reception_process_descriptor_t * | p_ts_descriptor_array | TS descriptor array that use to create queue. |
| layer3_switch_rx_timestamp_storage_t | rx_timestamp_storage | Configure RX timestamp storage. |
| uint32_t | array_length | Length of descriptor array. This length includes terminate descriptor at the end. |
| uint32_t | ports | Bitmap of ports that use this queue. |

### ◆ layer3_switch_cbs_cfg_t

| struct layer3_switch_cbs_cfg_t | | |
|---|---|---|
| Configuration of Credit Based Shaper. | | |
| Data Fields | | |
| uint8_t | band_width_list[8] | CBS band width [%] of each queue. |
| uint8_t | max_burst_num_list[8] | Maximum burst frame number of each queue. |

### ◆ layer3_switch_port_cfg_t

| struct layer3_switch_port_cfg_t | |
|---|---|
| Configuration of each Ethernet port. | |
| **Data Fields** | |
| uint8_t * | p_mac_address |

| | |
|---|---|
| | Pointer to MAC address. |
| | |
| bool | forwarding_to_cpu_enable |
| | Enable or disable reception on CPU. |
| | |
| layer3_switch_cbs_cfg_t * | p_cbs_cfg |
| | Pointer to CBS configuration. |
| | |
| void(* | p_callback )(ether_switch_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| ether_switch_callback_args_t * | p_callback_memory |
| | Pointer to optional callback argument memory. |
| | |
| void * | p_context |
| | Pointer to context to be passed into callback function. |
| | |

### ◆ layer3_switch_descriptor_queue_status_t

struct layer3_switch_descriptor_queue_status_t

Status of a descriptor queue.

Data Fields

| | | |
|---|---|---|
| uint32_t | head | Index at the head of the queue. This used for GetDescriptor API. |
| uint32_t | tail | Index at the tail of the queue. This used for SetDescriptor API. |
| bool | created | This queue is already created. |
| layer3_switch_descriptor_queue_cfg_t const * | p_queue_cfg | Configuration of this queue. |
| bool | rx_available | Indicates whether the queue is available for reception. |

### ◆ layer3_switch_ipv6_filter_address_offset_t

| struct layer3_switch_ipv6_filter_address_offset_t | | |
|---|---|---|
| IP address offset for creating IPv6 filter of Layer3 forwarding. | | |
| Data Fields | | |
| uint8_t | offset | Offset of IPv6 address. |
| layer3_switch_ip_address_direction_t | direction | Select source or destination. |

### ◆ layer3_switch_l3_stream_filter_cfg_t

| struct layer3_switch_l3_stream_filter_cfg_t | | |
|---|---|---|
| Configuration of stream filter in Layer3 forwarding. | | |
| Data Fields | | |
| uint16_t | filter_field_bitmask | Bitmask of which feilds is enabled in stream filter. Use layer3_switch_l3_filter_bitmask_t. |
| layer3_switch_ipv6_filter_address_offset_t | ipv6_address0 | Offset of IPv6 address 0. |
| layer3_switch_ipv6_filter_address_offset_t | ipv6_address1 | Offset of IPv6 address 1. |

### ◆ layer3_switch_frame_vlan_tag_t

| struct layer3_switch_frame_vlan_tag_t | | |
|---|---|---|
| VLAN tag structure. | | |
| Data Fields | | |
| uint16_t | pcp: 3 | Priority Code Point (3 bit). |
| uint16_t | dei: 1 | Drop Eligible Indicator (1 bit). |
| uint16_t | id: 12 | VLAN ID (12 bit). |

### ◆ layer3_switch_frame_filter_t

| struct layer3_switch_frame_filter_t | | |
|---|---|---|
| Frame filter of a MAC/VLAN/Layer3 forwarding entry. Set values for members required for filtering. | | |
| Data Fields | | |
| layer3_switch_table_entry_type_t | entry_type | Type of this entry. Select MAC, VLAN or Layer3. |
| uint8_t * | p_destination_mac_address | Destination MAC address. |
| uint8_t * | p_source_mac_address | Source MAC address. |
| layer3_switch_frame_vlan_tag_t | vlan_c_tag | VLAN C-TAG. |
| layer3_switch_frame_vlan_tag_t | vlan_s_tag | VLAN S-TAG. |

| | | |
|---|---|---|
| layer3_switch_ip_version_t | ip_version | IP version. Select IPv4, IPv6, or not IP packet. |
| uint8_t | protocol | IP protocol. |
| uint8_t * | p_source_ip_address | Source IP address. |
| uint8_t * | p_destination_ip_address | Destination IP address. |
| uint16_t | layer4_source_port | TCP/UDP source port. |
| uint16_t | layer4_destination_port | TCP/UDP destination port. |

### ◆ layer3_switch_l3_filter_t

| struct layer3_switch_l3_filter_t | | |
|---|---|---|
| Store filter information of Layer3 forwarding entry. | | |
| Data Fields | | |
| layer3_switch_frame_filter_t | frame | Target frame format that used to Layer3 forwarding. |

### ◆ layer3_switch_l3_update_config_t

| struct layer3_switch_l3_update_config_t | | |
|---|---|---|
| Configuration of a L2/L3 update feature for output frames. | | |
| Data Fields | | |
| uint32_t | enable_destination_ports | Destination ports that this update config is enabled. |
| uint32_t | update_field_bitmask | Bit mask of which fields will be updated. Use layer3_switch_l3_update_bitmask_t. |
| uint8_t * | p_mac_destination_address | MAC Destination Address. |
| layer3_switch_frame_vlan_tag_t | vlan_c_tag | VLAN C-tag. |
| layer3_switch_frame_vlan_tag_t | vlan_s_tag | VLAN S-tag. |
| layer3_switch_forwarding_r_tag_t | r_tag_update_mode | R-TAG update setting. |
| bool | sequence_number_generation_enable | Flags whether sequence number generation (FRER) is enabled for this entry. |

### ◆ layer3_switch_table_entry_cfg_t

| struct layer3_switch_table_entry_cfg_t | | |
|---|---|---|
| Table entry configuration of MAC/VLAN/Layer3 forwarding. | | |
| Data Fields | | |
| bool | entry_enable | Enable or disable entry. If this field is false, entry will be |

| | | |
|---|---|---|
| | | removed. |
| bool | security_enable | Entry is secure or not. |
| layer3_switch_frer_entry_cfg_t * | p_frer_entry_cfg | Configuration of the FRER entry(individual recovery) for this L3 entry (set to NULL : FRER is not valid in this L3 entry). |
| uint32_t | destination_ports | Destination ports of forwarding. |
| uint32_t | source_ports | Source ports that enable forwarding of incoming frame. |
| uint32_t | destination_queue_index | Destination queue. This fields is only used when a destination port is CPU. |
| uint32_t | internal_priority_update_enable | Enable to update internal priority. |
| uint32_t | internal_priority_update_value | Internal priority when updating is enabled. |
| union layer3_switch_table_entry_cfg_t | __unnamed__ | |

### ◆ layer3_switch_table_entry_t

| struct layer3_switch_table_entry_t | | |
|---|---|---|
| Table entry of MAC/VLAN/Layer3 forwarding. | | |
| Data Fields | | |
| layer3_switch_frame_filter_t | target_frame | Target frame of forwarding by this entry. |
| layer3_switch_table_entry_cfg_t | entry_cfg | Configuration of this entry. |

### ◆ layer3_switch_forwarding_port_cfg_t

| struct layer3_switch_forwarding_port_cfg_t | | |
|---|---|---|
| Configuration of forwarding feature for each port. | | |
| Data Fields | | |
| bool | mac_table_enable | Enable MAC table and forwarding feature. |
| bool | mac_reject_unknown | Reject frame with unknown MAC address. |
| bool | mac_hardware_learning_enable | Enable hardware learning and migration. |
| bool | vlan_table_enable | Enable VLAN table and forwarding feature. |
| bool | vlan_reject_unknown | Reject frame with unknown VLAN ID. |

| | | |
|---|---|---|
| layer3_switch_vlan_ingress_mode_t | vlan_ingress_mode | Select Tag-based VLAN or Port-based VLAN for incoming frame. |
| layer3_switch_vlan_egress_mode_t | vlan_egress_mode | Tagging/untagging mode for outgoing frame. |
| layer3_switch_frame_vlan_tag_t | vlan_s_tag | S-TAG of this port. When egress mode is hardware SC-TAG, add this to outgoing frame. |
| layer3_switch_frame_vlan_tag_t | vlan_c_tag | C-TAG of this port. When egress mode is hardware C-TAG, add this to outgoing frame. |
| bool | layer3_table_enable | Enable Layer3 table and forwarding feature. |
| bool | layer3_reject_unknown | Reject frame that not found in Layer3 table. |
| bool | layer3_ipv4_filter_enable | Enable IPv4 stream filter. |
| bool | layer3_ipv6_filter_enable | Enable IPv6 stream filter. |
| bool | layer3_l2_filter_enable | Enable L2 stream filter. |

### ◆ layer3_switch_table_t

| struct layer3_switch_table_t | | |
|---|---|---|
| Forwarding table containing MAC/VLAN/Layer3 forwarding entries. | | |
| Data Fields | | |
| layer3_switch_table_entry_t * | p_mac_entry_list | List of MAC entries. |
| uint32_t | mac_list_length | Length of the MAC entry list. |
| layer3_switch_table_entry_t * | p_vlan_entry_list | List of VLAN entries. |
| uint32_t | vlan_list_length | Length of the VLAN entry list. |
| layer3_switch_table_entry_t * | p_l3_entry_list | List of Layer3 entries. |
| uint32_t | l3_list_length | Length of the Layer3 entry list. |

### ◆ layer3_switch_table_cfg_t

| struct layer3_switch_table_cfg_t | | |
|---|---|---|
| Configuration of the forwarding table. | | |
| Data Fields | | |
| layer3_switch_table_t * | p_table | Pointer to forwarding table. |
| layer3_switch_forwarding_port_cfg_t | port_cfg_list[BSP_FEATURE_ETHER_NUM_CHANNELS+1] | Forwarding configuration of each port. |
| uint32_t | unsecure_entry_maximum_num | Maximum number of unsecure entries. |
| | | |

| | | |
|---|---|---|
| bool | mac_entry_aging_enable | Enable aging feature of MAC table. |
| uint32_t | mac_entry_aging_time_sec | Time[s] to delete an entry by aging. |
| layer3_switch_vlan_mode_t | vlan_mode | VLAN mode options: NO VLAN, C-TAG, or SC-TAG. |
| layer3_switch_l3_stream_filter_cfg_t | l3_stream_filter_cfg | Configuration of stream filter in Layer3 forwarding. |
| layer3_switch_frer_cfg_t | frer_cfg | Configuration of FRER feature. |

### ◆ layer3_switch_extended_cfg_t

| struct layer3_switch_extended_cfg_t | | |
|---|---|---|
| ESWM extension configures each Ethernet port and forwarding feature. | | |
| Data Fields | | |
| ether_phy_instance_t const * | p_ether_phy_instances[BSP_FEATURE_ETHER_NUM_CHANNELS] | List of pointers to ETHER_PHY instance. |
| gptp_instance_t const * | p_gptp_instance | Pointer to a gPTP instance. |
| uint32_t | fowarding_target_port_masks[BSP_FEATURE_ETHER_NUM_CHANNELS] | List of ports to which incoming frames are forwarded. |
| uint8_t * | p_mac_addresses[BSP_FEATURE_ETHER_NUM_CHANNELS] | |
| uint32_t | ipv_queue_depth_list[BSP_FEATURE_ETHER_NUM_CHANNELS][8] | List of IPV queue depth for each port. |
| layer3_switch_l3_filter_t * | l3_filter_list | Filter list of layer3 routing. |
| uint32_t | l3_filter_list_length | Length of Layer3 filter list. |
| layer3_switch_port_cfg_t * | p_port_cfg_list[BSP_FEATURE_ETHER_NUM_CHANNELS] | Configuration for each port. |
| IRQn_Type | etha_error_irq_port_0 | ETHA error interrupt number for port 0. |
| IRQn_Type | etha_error_irq_port_1 | ETHA error interrupt number for port 1. |
| uint8_t | etha_error_ipl_port_0 | ETHA error interrupt priority for port 0. |
| uint8_t | etha_error_ipl_port_1 | ETHA error interrupt priority for port 1. |
| uint8_t | gptp_timer_numbers[BSP_FEATURE_ESWM_GPTP_TIMER_NUM] | List of timer numbers for transmission/reception timestamp. |

### ◆ layer3_switch_instance_ctrl_t

| struct layer3_switch_instance_ctrl_t | |
|---|---|
| LAYER3_SWITCH control block. DO NOT INITIALIZE. Initialization occurs when ether_switch_api_t::open is called. | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| ether_switch_cfg_t const * | p_cfg |
| | Pointer to initial configurations. |
| | |
| R_GWCA0_Type * | p_gwca_reg |
| | Pointer to GWCA register. |
| | |
| uint32_t | allocated_descriptor_queue_index |
| | Index of the descriptor pool. |
| | |
| layer3_switch_basic_descriptor_t | p_descriptor_queue_list [LAYER3_SWITCH_CFG_AVAILABLE_QUEUE_NUM] |
| | Descriptor queue lists used by hardware. |
| | |
| layer3_switch_descriptor_queue_status_t | p_queues_status [LAYER3_SWITCH_CFG_AVAILABLE_QUEUE_NUM] |
| | Status of each descriptor queues. |
| | |
| layer3_switch_port_cfg_t | p_port_cfg_list [BSP_FEATURE_ETHER_NUM_CHANNELS] |
| | Configuration for each port. |
| | |
| layer3_switch_table_status_t | table_status |
| | Forwarding table is initialized or not. |
| | |

| | |
|---|---|
| uint32_t | l3_entry_count |
| | Counts of valid LAYER3 entry. |
| | |
| uint8_t | l3_routing_number |
| | Routing number for L2/L3 update feature. |
| | |
| uint8_t | l3_remapping_number |
| | Remapping number for L2/L3 update feature. |
| | |
| layer3_switch_ts_descriptor_queue_status_t | ts_descriptor_queue_status_list [BSP_FEATURE_ESWM_TS_DESCRIPTOR_QUEUE_MAX_NUM] |
| | Status of TS reception descriptor queues. |
| | |
| uint32_t | valid_frer_entry_num |
| | Number of valid FRER entry. |
| | |
| layer3_switch_frer_sequence_recovery_status_t | frer_sequence_recovery_status [BSP_FEATURE_ESWM_FRER_TABLE_SIZE] |
| | Status of table for each FRER sequence recovery entry. |
| | |
| uint32_t | used_frer_sequence_generator_num |
| | Number of the sequence number generator. |
| | |
| void(* | p_callback )(ether_switch_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| ether_switch_callback_args_t * | p_callback_memory |
| | Pointer to optional callback argument memory. |

| | |
|---|---|
| void * | p_context |
| | Pointer to context to be passed into callback function. |

### ◆ layer3_switch_tas_entry_t

| struct layer3_switch_tas_entry_t | | |
|---|---|---|
| Configuration of the gate operation. | | |
| Data Fields | | |
| layer3_switch_tas_gate_state_t | state | Gate state. |
| uint32_t | time | Time associated with the entry gate state [nsec]. |

### ◆ layer3_switch_tas_gate_cfg_t

| struct layer3_switch_tas_gate_cfg_t | | |
|---|---|---|
| Configuration of the gate. | | |
| Data Fields | | |
| layer3_switch_tas_gate_state_t | initial_gate_state | Initial gate state when the cycle starts. |
| uint8_t | tas_entry_num | Number of TAS entries included in this gate. |
| layer3_switch_tas_entry_t * | p_tas_entry_list | List of TAS entries included in this gate. |

### ◆ layer3_switch_tas_cfg_t

| struct layer3_switch_tas_cfg_t | | |
|---|---|---|
| Configuration of the TAS. | | |
| Data Fields | | |
| uint8_t | gptp_timer_number | gPTP timer number. |
| uint32_t | cycle_time_start_high | Upper 32 bits of TAS cycle start time [nsec]. |
| uint32_t | cycle_time_start_low | Lower 32 bits of TAS cycle start time [nsec]. |
| uint32_t | cycle_time | TAS cycle time [nsec]. |
| layer3_switch_tas_gate_cfg_t | gate_cfg_list[8] | List of TAS gate configurations. |

## Enumeration Type Documentation

### ◆ layer3_switch_descriptor_type_t

| enum layer3_switch_descriptor_type_t | |
|---|---|
| GWCA descriptor type. | |
| Enumerator | |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_LINKFIX | Linkfix. Control element pointing to next descriptor in chain. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_IS | Frame Empty Incremental Start. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_IC | Frame Empty Incremental Continue. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_ND | Frame Empty No Data Storage. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY | Frame Empty. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_START | Frame Empty Start. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_MID | Frame Empty Mid. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEMPTY_END | Frame Empty End. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FSINGLE | Frame Single. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FSTART | Frame Start. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FMID | Frame Mid. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_FEND | Frame End. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_LEMPTY | Link Empty. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_EEMPTY | EOS Empty. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_LINK | Link. |
| LAYER3_SWITCH_DESCRIPTOR_TYPE_EOS | End of Set. |

### ◆ layer3_switch_port_bitmask_t

| enum layer3_switch_port_bitmask_t | |
|---|---|
| Bitmasks for each port. | |
| Enumerator | |
| LAYER3_SWITCH_PORT_BITMASK_PORT0 | Port 0. |
| LAYER3_SWITCH_PORT_BITMASK_PORT1 | Port 1. |
| LAYER3_SWITCH_PORT_BITMASK_PORT2 | Port 2. |

### ◆ layer3_switch_queue_type_t

| enum layer3_switch_queue_type_t | |
|---|---|
| Descriptor queue type. | |
| Enumerator | |
| LAYER3_SWITCH_QUEUE_TYPE_RX | Reception queue. |
| LAYER3_SWITCH_QUEUE_TYPE_TX | Transmission queue. |

### ◆ layer3_switch_descriptor_format_t

| enum layer3_switch_descriptor_format_t | |
|---|---|
| Descriptor format type. | |
| Enumerator | |
| LAYER3_SWITCH_DISCRIPTOR_FORMTAT_BASIC | Using basic descriptor. |
| LAYER3_SWITCH_DISCRIPTOR_FORMTAT_EXTENDED | Using extended descriptor with additional fields. |
| LAYER3_SWITCH_DISCRIPTOR_FORMTAT_TX_TIMESTAMP | Using TX timestamp descriptor. |

### ◆ layer3_switch_write_back_mode_t

| enum layer3_switch_write_back_mode_t | |
|---|---|
| Write back mode. | |
| Enumerator | |
| LAYER3_SWITCH_WRITE_BACK_MODE_FULL | All fields are updated by hardware. |
| LAYER3_SWITCH_WRITE_BACK_MODE_DISABLE | No fields are updated by hardware. |
| LAYER3_SWITCH_WRITE_BACK_MODE_KEEP_DT | Fields exclude DT are updated by hardware. |

### ◆ layer3_switch_ip_version_t

| enum layer3_switch_ip_version_t | |
|---|---|
| IP version options. | |
| Enumerator | |
| LAYER3_SWITCH_IP_VERSION_NONE | Not IP packet. |
| LAYER3_SWITCH_IP_VERSION_IPV4 | IPv4 packet. |
| LAYER3_SWITCH_IP_VERSION_IPV6 | IPv6 packet. |

### ◆ layer3_switch_ip_address_direction_t

| enum layer3_switch_ip_address_direction_t | |
|---|---|
| Direction of IP address. | |
| Enumerator | |
| LAYER3_SWITCH_IP_ADDRESS_DIRECTION_SOURCE | Use IP source address in stream ID. |
| LAYER3_SWITCH_IP_ADDRESS_DIRECTION_DESTINATION | Use IP destination address in stream ID. |

### ◆ layer3_switch_magic_packet_detection_t

| enum layer3_switch_magic_packet_detection_t | |
|---|---|
| Magic packet detection options. | |
| Enumerator | |
| LAYER3_SWITCH_MAGIC_PACKET_DETECTION_ENABLE | Enable magic packet detection. |
| LAYER3_SWITCH_MAGIC_PACKET_DETECTION_DISABLE | Disable magic packet detection. |

### ◆ layer3_switch_vlan_mode_t

| enum layer3_switch_vlan_mode_t | |
|---|---|
| VLAN detection mode. | |
| Enumerator | |
| LAYER3_SWITCH_VLAN_MODE_NO_VLAN | Not use VLAN feature. |
| LAYER3_SWITCH_VLAN_MODE_C_TAG | Detect and use VLAN C-TAG. |
| LAYER3_SWITCH_VLAN_MODE_SC_TAG | Detect and use VLAN SC-TAG. |

### ◆ layer3_switch_vlan_ingress_mode_t

| enum layer3_switch_vlan_ingress_mode_t | |
|---|---|
| VLAN ingress mode determines whether the switch operates with tagged VLAN or port-based VLAN. | |
| Enumerator | |
| LAYER3_SWITCH_VLAN_INGRESS_MODE_TAG_BASED | Use VLAN ID of the incoming frame. |
| LAYER3_SWITCH_VLAN_INGRESS_MODE_PORT_BASED | Use VLAN ID of the incoming port. |

### ◆ layer3_switch_vlan_egress_mode_t

| enum layer3_switch_vlan_egress_mode_t | |
|---|---|
| VLAN egress mode determines the VLAN tag that is added to output frames. | |
| Enumerator | |
| LAYER3_SWITCH_VLAN_EGRESS_MODE_NO_VLAN | Frame always outgoing without VLAN TAG. |
| LAYER3_SWITCH_VLAN_EGRESS_MODE_C_TAG | Frame outgoing with C-TAG of incoming frame. |
| LAYER3_SWITCH_VLAN_EGRESS_MODE_HW_C_TAG | Frame always outgoing with C-TAG of output port. |
| LAYER3_SWITCH_VLAN_EGRESS_MODE_SC_TAG | Frame always outgoing with SC-TAG of incoming frame. |
| LAYER3_SWITCH_VLAN_EGRESS_MODE_HW_SC_TAG | Frame always outgoing with SC-TAG of output port. |

### ◆ layer3_switch_ip_protocol_t

| enum layer3_switch_ip_protocol_t | |
|---|---|
| IP protocols type. | |
| Enumerator | |
| LAYER3_SWITCH_IP_PROTOCOL_TCP | IP protocol is TCP. |
| LAYER3_SWITCH_IP_PROTOCOL_UDP | IP protocol is UDP. |

### ◆ layer3_switch_forwarding_r_tag_t

| enum layer3_switch_forwarding_r_tag_t | |
|---|---|
| R-TAG filed update options. | |
| Enumerator | |
| LAYER3_SWITCH_FORWARDING_R_TAG_UPDATE | Incoming frame with R-TAG will be updated with new R-TAG. |
| LAYER3_SWITCH_FORWARDING_R_TAG_ALWAYS_ADD | ALl incoming frame will be added new R-TAG. |
| LAYER3_SWITCH_FORWARDING_R_TAG_ALWAYS_ELIMINATE | ALl incoming frame will be removed R-TAG. |

### ◆ layer3_switch_l3_filter_bitmask_t

| enum layer3_switch_l3_filter_bitmask_t | |
|---|---|
| Frame filters options for layer3 forwarding. | |
| Enumerator | |
| LAYER3_SWITCH_L3_FILTER_BITMASK_MAC_DESTINATION | Enable MAC destination address. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_MAC_SOURCE | Enable MAC source address. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_ID | Enable VLAN ID of S-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_PCP | Enable PCP of S-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_STAG_DEI | Enable DEI of S-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_ID | Enable VLAN ID of C-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_PCP | Enable PCP of C-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_CTAG_DEI | Enable DEI of C-TAG. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_IP_SOURCE_ADDRESS | Enable IP source address. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_IP_DESTINATION_ADDRESS | Enable IP destination address. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_PROTOCOL | Enable IP protocol. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_IP_SOURCE_PORT | Enable layer4 source port. |
| LAYER3_SWITCH_L3_FILTER_BITMASK_IP_DESTINATION_PORT | Enable layer4 destination port. |

### ◆ layer3_switch_l3_update_bitmask_t

| enum layer3_switch_l3_update_bitmask_t | |
|---|---|
| Enable L2/L3 update feature. | |
| Enumerator | |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_TTL | Update TTL. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_MAC_DESTINATION | Update MAC destination address. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_MAC_SOURCE | Update MAC source address. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_ID | Update VLAN ID of C-TAG. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_PCP | Update PCP of C-TAG. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_CTAG_DEI | Update DEI of C-TAG. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_ID | Update VLAN ID of C-TAG. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_PCP | Update PCP of C-TAG. |
| LAYER3_SWITCH_L3_UPDATE_BITMASK_STAG_DEI | Update DEI of C-TAG. |

### ◆ layer3_switch_table_entry_type_t

| enum layer3_switch_table_entry_type_t | |
|---|---|
| Table entry type for forwarding feature. | |
| Enumerator | |
| LAYER3_SWITCH_TABLE_ENTRY_TYPE_EMPTY | Entry is not initialized yet. |
| LAYER3_SWITCH_TABLE_ENTRY_TYPE_MAC | Entry of MAC table. |
| LAYER3_SWITCH_TABLE_ENTRY_TYPE_VLAN | Entry of VLAN table. |
| LAYER3_SWITCH_TABLE_ENTRY_TYPE_LAYER3 | Entry of Layer3 table. |

### ◆ layer3_switch_table_status_t

| enum layer3_switch_table_status_t | |
|---|---|
| Table status for forwarding feature. | |
| Enumerator | |
| LAYER3_SWITCH_TABLE_STATUS_UNINITIALIZED | Forwarding table is uninitialized. |
| LAYER3_SWITCH_TABLE_STATUS_INITIALIZED | Forwarding table is initialized. |

### ◆ layer3_switch_tas_gate_state_t

| enum layer3_switch_tas_gate_state_t | |
|---|---|
| TAS gate state. | |
| Enumerator | |
| LAYER3_SWITCH_TAS_GATE_STATE_CLOSE | Gate is closed. |
| LAYER3_SWITCH_TAS_GATE_STATE_OPEN | Gate is opened. |

### ◆ layer3_switch_rx_timestamp_storage_t

| enum layer3_switch_rx_timestamp_storage_t | |
|---|---|
| Enable or disable rx timestamp storage. | |
| Enumerator | |
| LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_DISABLE | Timestamp is not added in the descriptor. |
| LAYER3_SWITCH_RX_TIMESTAMP_STORAGE_ENABLE | Timestamp is added in the descriptor. |

### ◆ layer3_switch_ts_descriptor_queue_status_t

| enum layer3_switch_ts_descriptor_queue_status_t | |
|---|---|
| Used or not timestamp descriptor queue. | |
| Enumerator | |
| LAYER3_SWITCH_TS_DESCRIPTOR_QUEUE_STATUS_UNUSED | TS descriptor queue is unused. |
| LAYER3_SWITCH_TS_DESCRIPTOR_QUEUE_STATUS_USED | TS descriptor queue is used. |

## Function Documentation

### ◆ R_LAYER3_SWITCH_Open()

fsp_err_t R_LAYER3_SWITCH_Open ( ether_switch_ctrl_t *const *p_ctrl*, ether_switch_cfg_t const *const *p_cfg* )

Initializes the switch module and applies configurations. Implements ether_switch_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to control block, config structure or extended config structure is NULL. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |
| FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION | Initialization of PHY-LSI failed. |
| FSP_ERR_INVALID_ARGUMENT | Invalid configuration value. |

### ◆ R_LAYER3_SWITCH_Close()

fsp_err_t R_LAYER3_SWITCH_Close ( ether_switch_ctrl_t *const *p_ctrl*)

Disables interrupts and stop module. Implements ether_switch_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | Control block is not open. |

### ◆ R_LAYER3_SWITCH_CreateDescriptorQueue()

fsp_err_t R_LAYER3_SWITCH_CreateDescriptorQueue ( ether_switch_ctrl_t *const *p_ctrl*, uint32_t *const *p_queue_index*, const layer3_switch_descriptor_queue_cfg_t *const *p_queue_cfg* )

Create a new descriptor queue and set it to LINKFIX table. This function must be called before calling R_LAYER3_SWITCH_SetDescriptor and R_LAYER3_SWITCH_GetDescriptor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Descriptor created successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | Control block is not open. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_OUT_OF_MEMORY | Descriptor queue list is depleted. |
| FSP_ERR_OVERFLOW | TS descriptor queue is used. |

### ◆ R_LAYER3_SWITCH_SetDescriptor()

fsp_err_t R_LAYER3_SWITCH_SetDescriptor ( ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index*, layer3_switch_descriptor_t const *const *p_descriptor* )

Set descriptor data to a target descriptor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Descriptor set successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | Control block is not open. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Queue index is invalid. |
| FSP_ERR_NOT_INITIALIZED | This descriptor queue is not created. |
| FSP_ERR_IN_USE | Target descriptor is now running. |
| FSP_ERR_OVERFLOW | Descriptor queue is full. |

### ◆ R_LAYER3_SWITCH_GetDescriptor()

fsp_err_t R_LAYER3_SWITCH_GetDescriptor ( ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index*, layer3_switch_descriptor_t *const *p_descriptor* )

Get descriptor data from a target descriptor.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Descriptor got successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | Control block is not open. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Queue index is invalid. |
| FSP_ERR_NOT_INITIALIZED | This descriptor queue is not created or target descriptor is not set. |
| FSP_ERR_IN_USE | Target descriptor is now running. |

### ◆ R_LAYER3_SWITCH_StartDescriptorQueue()

fsp_err_t R_LAYER3_SWITCH_StartDescriptorQueue ( ether_switch_ctrl_t *const *p_ctrl*, uint32_t *queue_index* )

Reload and enable a descriptor queue. In a TX descriptor queue, the queue start transmission. In a RX descriptor queue, the queue start reception.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Descriptor queue started successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | Control block is not open. |
| FSP_ERR_INVALID_ARGUMENT | Queue index is invalid. |
| FSP_ERR_NOT_INITIALIZED | Target descriptor queue is not created. |
| FSP_ERR_IN_USE | Target descriptor queue is already running. |
| FSP_ERR_INVALID_DATA | Target TX queue have no data. |

### ◆ R_LAYER3_SWITCH_CallbackSet()

fsp_err_t R_LAYER3_SWITCH_CallbackSet ( ether_switch_ctrl_t *const *p_ctrl*, void(*)(ether_switch_callback_args_t *) *p_callback*, void *const *p_context*, ether_switch_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_LAYER3_SWITCH_ConfigurePort()

fsp_err_t R_LAYER3_SWITCH_ConfigurePort ( ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port*, layer3_switch_port_cfg_t * *p_port_cfg* )

Configure Ethernet port features, including callback function for each port.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port configured successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Port number is invalid. |

### ◆ R_LAYER3_SWITCH_AddTableEntry()

fsp_err_t R_LAYER3_SWITCH_AddTableEntry ( ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_frame_filter_t const *const *p_target_frame*, layer3_switch_table_entry_cfg_t const *const *p_entry_cfg* )

Add or update an entry of the forwarding table.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully add/updated an entry into the table. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_OVERFLOW | The number of unsecure entries exceeded the configured value. |
| FSP_ERR_WRITE_FAILED | A hardware error occurred while learning the entry. |
| FSP_ERR_INVALID_ARGUMENT | Target frame or entry type is invalid. |
| FSP_ERR_INVALID_MODE | VLAN feature is disabled and a VLAN entry is passed. |

### ◆ R_LAYER3_SWITCH_SearchTableEntry()

fsp_err_t R_LAYER3_SWITCH_SearchTableEntry ( ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_frame_filter_t const *const *p_target_frame*, layer3_switch_table_entry_cfg_t *const *p_entry_cfg* )

Search an entry from the forwarding table.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The entry is found successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_NOT_FOUND | The entry is not found in the table. |
| FSP_ERR_INVALID_ARGUMENT | Target frame or entry type is invalid. |
| FSP_ERR_INVALID_MODE | VLAN feature is disabled and a VLAN entry is passed. |

### ◆ R_LAYER3_SWITCH_ConfigureTable()

fsp_err_t R_LAYER3_SWITCH_ConfigureTable ( ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_table_cfg_t const *const *p_table_cfg* )

Configure and initialize an forwarding table.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Table configured successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_WRITE_FAILED | Failed to add entries. |

### ◆ R_LAYER3_SWITCH_GetTable()

fsp_err_t R_LAYER3_SWITCH_GetTable ( ether_switch_ctrl_t *const *p_ctrl*, layer3_switch_table_t *const *p_table* )

Get a pointer to the forwarding table.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Table got successfully. |
| FSP_ERR_ASSERTION | Pointer to control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_INVALID_MODE | VLAN feature is disabled and a VLAN entry list is passed. |

### ◆ R_LAYER3_SWITCH_ConfigureTAS()

fsp_err_t R_LAYER3_SWITCH_ConfigureTAS ( ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port*, layer3_switch_tas_cfg_t * *p_tas_cfg* )

Configure Time Aware Shaper feature.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | TAS configure successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_POINTER | Pointer to a argument is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Port number is invalid. |
| FSP_ERR_UNSUPPORTED | TAS feature is not enabled in the configuration. |

### ◆ R_LAYER3_SWITCH_EnableTAS()

fsp_err_t R_LAYER3_SWITCH_EnableTAS ( ether_switch_ctrl_t *const *p_ctrl*, uint8_t *port* )

Enable Time Aware Shaper feature.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | TAS enabled successfully. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Port number is invalid. |
| FSP_ERR_UNSUPPORTED | TAS feature is not enabled in the configuration. |

## 5.2.12.24 NFC Reader Driver on PTX (rm_nfc_reader_ptx)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | RM_NFC_READER_PTX_Open (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, nfc_reader_ptx_cfg_t const *const p_cfg) |
| fsp_err_t | RM_NFC_READER_PTX_DiscoveryStart (nfc_reader_ptx_instance_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_NFC_READER_PTX_StatusGet (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, ptxIoTRd_StatusType_t status_type, uint8_t *const |

| | |
|---|---|
| | p_status) |
| fsp_err_t | RM_NFC_READER_PTX_CardRegistryGet (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, ptxIoTRd_CardRegistry_t **pp_card_registry) |
| fsp_err_t | RM_NFC_READER_PTX_CardActivate (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, ptxIoTRd_CardParams_t *const p_card_params, ptxIoTRd_CardProtocol_t protocol) |
| fsp_err_t | RM_NFC_READER_PTX_DataExchange (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, nfc_reader_ptx_data_info_t *const p_data_info) |
| fsp_err_t | RM_NFC_READER_PTX_ReaderDeactivation (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, nfc_reader_ptx_return_state_t return_state) |
| fsp_err_t | RM_NFC_READER_PTX_PowerModeSet (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, nfc_reader_ptx_power_mode_t power_mode) |
| fsp_err_t | RM_NFC_READER_PTX_NDEF_Init (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, nfc_reader_ptx_data_info_t *const p_data_info) |
| fsp_err_t | RM_NFC_READER_PTX_NDEF_DataExchange (nfc_reader_ptx_instance_ctrl_t *const p_ctrl, uint8_t *p_ndef_message, uint32_t message_length, nfc_reader_ptx_ndef_read_write_t ndef_read_write_flag) |
| fsp_err_t | RM_NFC_READER_PTX_NDEF_Lock (nfc_reader_ptx_instance_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_NFC_READER_PTX_Close (nfc_reader_ptx_instance_ctrl_t *const p_ctrl) |

### Detailed Description

NFC Reader driver implementation using the PTX NFC module on RA MCUs.

# Overview

This middleware module supplies an implementation for the NFC IoT Reader on the PTX module. The NFC functionalities are provided via a modular NFC Soft Controller (NCS) architecture, and this FSP module will include the NSC (aka NFC SDK). Functionality is provided in a split-stack architecture, where time-critical operations are running on the on-chip MCU, and the rest of the NFC logic is in the host controller to carry out applications such as IoT reader. The driver may also be adapted to further support next-generation PTX devices in future (which have not yet been released).

The NFC Reader module supports the following modules:

- PTX105R (https://www.renesas.com/en/products/wireless-connectivity/nfc/ptx105r-mid-power-multi-protocol-nfc-forum-compliant-reader)

**Features**

The NFC Reader PTX driver supplies these features:

- Initializes the IOTRD (IoT Reader) API and NSC stack
- Initializes the PTX chip
- Discovers cards according to NFC Forum
- Selects a specific card in case multiple cards/protocols were discovered
- Retrieves card details like technical/activation parameters
- Exchanges RF data and bitstreams
- Stops RF communication
- NDEF add-on API for exchanging data using the NDEF (NFC Data Exchange Format) standard
- Discovery support for "Low-Power-Card-Detection"
- Optionally puts the PTX chip into and wake up from stand-by mode

Future support is considered for the following features:

- Optionally updates RF and System configuration parameters at runtime
- Controls various GPIO pins of the PTX chip
- Plus several add-on features for additional operations and testing

# Configuration

**Build Time Configurations for rm_nfc_reader_ptx**

The following build time configurations are defined in fsp_cfg/rm_nfc_reader_ptx_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Add-on | | | |
| NDEF Support | • Disabled<br>• Enabled | Disabled | Includes the APIs provided by the NDEF Add-on. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| NFC NDEF Work Buffer Size | Must be an integer greater than 0 | 64 | Size in bytes of the NFC NDEF work buffer used for internally exchanging data with a tag. Must be an integer greater than 0. |

**Configurations for Networking > NFC IoT Reader on PTX (rm_nfc_reader_ptx)**

This module can be added to the Stacks tab via New Stack > Networking > NFC IoT Reader on PTX (rm_nfc_reader_ptx).

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| NFC Poll Type | | | |
| Type-A | • Disabled<br>• Enabled | Enabled | Select to discover NFC Type-A tags |
| Type-B | • Disabled<br>• Enabled | Enabled | Select to discover NFC Type-B tags |
| Type-V | • Disabled<br>• Enabled | Enabled | Select to discover NFC Type-V tags |
| Type-F | • Disabled<br>• Enabled | Enabled | Select to discover NFC Type-F tags |
| Temperature Sensor | | | |
| Calibrate Sensor | • Disabled<br>• Enabled | Enabled | Enable/Disable on-board temperature sensor calibration. Must be executed once per chip (lifetime), it is not needed after. |
| Shutdown Temperature (°C) | Must be an integer greater than 0 and less than 223 | 100 | Set the expected thermal shutdown threshold value (Celsius). |
| Ambient Temperature (°C) | Must be an integer greater than 0 and less than 85 | 25 | Set the ambient temperature value at which calibration takes place (Celsius). |
| Name | Name must be a valid C symbol | g_nfc_reader_ptx0 | Module name. |
| Set Polling Mode | Must be an integer greater than or equal to 0 and less than 256 | 0 | Set polling mode, where 0 is Regular Polling, 1 is Low-power Card Detection (LPCD), and 2-255 is LPCD with every n-th cycle as Regular Polling |
| NFC Interrupt Port | Refer to the RA Configuration tool for available options. | Disabled | Specify the interrupt port/pin used by the NFC chip to signal for SPI communications. |
| NFC Interrupt Pin | Refer to the RA Configuration tool for available options. | module.driver.nfc_reader_ptx.pin_control_pin.PIN_DISABLE | Specify the interrupt port/pin used by the NFC chip to signal for SPI communications. |
| Device Limit | Must be an integer greater than 0 and less than 50 | 5 | Select the max number of tags to discover. |

| | | | |
|---|---|---|---|
| Idle Time | Must be an integer greater than 0 and less than 65535 | 100 | Select idle time between polling cycles (ms). |

Note: This module provides the basic configurations in the FSP Configurator required to discover, activate, and exchange data with an NFC tag. The user can then copy these configurations to a new structure if they want to connect to a new tag type/make changes to the configurations dynamically.

**Interrupt Configuration**

Refer to SPI (r_sci_spi). R_SCI_SPI_Open() is called by this module

**Clock Configuration**

Refer to SPI (r_sci_spi).

**Pin Configuration**

Refer to SPI (r_sci_spi). R_SCI_SPI_Open() is called by this module

# Usage Notes

- For all information on the PTX NFC IoT Reader, including data types, APIs, the SDK, etc., refer to the PTX105R documentation on the webpage above.
- This module includes the NFC SDK which must be stored and compiled on the host RA MCU. The NFC IoT Reader claims that code size is roughly 39K B Code Flash (53 KB with the reference application) and 3-4 KB of Stack memory for the reference application. Consider this when looking at the device and the user application.
- This module also requires a minimal amount of heap allocation (256 bytes). Be sure to allocate some heap size in the FSP Configurator BSP tab.
- The NFC SDK provided with this module contains platform-specific and device-specific code related to using the FSP on RA devices. For other devices, the Platform API (ptxPLAT) must be implemented/ported.
- When selecting the NFC tag protocol (Type-A, Type-B, Type-V, and Type-F), ensure that at least one type is selected.
- For data exchange, the user must provide transmission/receive buffers with the exact data packet for the intended tag type. For information on commands, responses, and parameters, definitions can be found in the NFC Specification.
- The NFC SDK contains a default RF waveform configuration file (ptxNSC_RfConfigVal.c) which is compatible with the PTX105RQC Pmod board. For custom boards, this configuration file will need to be generated with the GUI tool available on the module webpage.
- The NFC chip provides an external interrupt pin which links to the ptxPLAT_GPIO_IsrCallback callback function. The same pin is also used during SPI communications for the NFC chip to signal it is ready and must be implemented in the FSP Configurator. The port/pin chosen for the interrupt pin should be the same for the NFC module properties and the External IRQ properties.

**Limitations**

- The module will only support SPI for the first release, with other protocols (I2C, UART) planned for future releases.
- The PTX105RQC Pmod board only supports SPI, so consider the evaluation kit (PTX105REK) for other interfaces.
- The NFC chip reset pin (labeled as SEN pin) isn't supported in the FSP currently, and it does

not have a pinout on the PTX105RQC Pmod board.

# Examples

### Basic Example

This is a basic example of minimal use of the NFC IoT Reader module in an application.

```
void nfc_basic_example (void)
{
 fsp_err_t                err       = FSP_SUCCESS;
    ptxIotRdInt_Demo_State_t demo_state = IoTRd_DemoState_WaitForActivation;
    err = RM_NFC_READER_PTX_Open(&g_rm_nfc_reader_ptx_instance,
&g_nfc_reader_ptx_cfg);
    assert(FSP_SUCCESS == err);
    err = RM_NFC_READER_PTX_DiscoveryStart(&g_rm_nfc_reader_ptx_instance);
    assert(FSP_SUCCESS == err);
    ptxIoTRd_CardRegistry_t * card_registry = NULL;
    err = RM_NFC_READER_PTX_CardRegistryGet(&g_rm_nfc_reader_ptx_instance,
&card_registry);
    assert(FSP_SUCCESS == err);
 /* T2T Protocol Example => READ BLOCK 0 */
 const uint8_t PROT_T2T_EXAMPLE[] = {0x30, 0x00};
    nfc_send_buf.tx_length = sizeof(PROT_T2T_EXAMPLE);
    memcpy(&nfc_tx_buf[0], &PROT_T2T_EXAMPLE[0], nfc_send_buf.tx_length);
 while (1)
    {
 /* Check regularly for critical system errors */
       uint8_t system_state = PTX_SYSTEM_STATUS_OK;
       err = RM_NFC_READER_PTX_StatusGet(&g_rm_nfc_reader_ptx_instance,
StatusType_System, &system_state);
      assert(FSP_SUCCESS == err);
 if ((PTX_SYSTEM_STATUS_ERR_OVERCURRENT == system_state) ||
(PTX_SYSTEM_STATUS_ERR_TEMPERATURE == system_state))
      {
 /* Critical error detected, reset the device and stop */
 RM_NFC_READER_PTX_Close(&g_rm_nfc_reader_ptx_instance);
```

```
 while (1)
      {
 /* Do nothing */
            }
            ;
       }
 /* Check optionally if PA current-limiter got activated */
       uint8_t last_rf_error = PTX_RF_ERROR_NTF_CODE_NO_ERROR;
       err = RM_NFC_READER_PTX_StatusGet(&g_rm_nfc_reader_ptx_instance,
StatusType_LastRFError, &last_rf_error);
      assert(FSP_SUCCESS == err);
 switch (demo_state)
      {
 case IoTRd_DemoState_WaitForActivation:
      {
                uint8_t discover_status = RF_DISCOVER_STATUS_NO_CARD;
                err = RM_NFC_READER_PTX_StatusGet(&g_rm_nfc_reader_ptx_instance,
StatusType_Discover, &discover_status);
      assert(FSP_SUCCESS == err);
 switch (discover_status)
      {
 case RF_DISCOVER_STATUS_NO_CARD:
      {
 /* Sleep */
 R_BSP_SoftwareDelay(NFC_READER_NO_CARD_SLEEP_TIME, BSP_DELAY_UNITS_MILLISECONDS);
 break;
              }
 case RF_DISCOVER_STATUS_CARD_ACTIVE:
      {
 /* Card activated */
 /* Get card details to find the protocol/decide what data to send */
      err = RM_NFC_READER_PTX_CardRegistryGet(&g_rm_nfc_reader_ptx_instance,
&card_registry);
      assert(FSP_SUCCESS == err);
```

```
                        demo_state = IoTRd_DemoState_DataExchange;
 break;
             }
 case RF_DISCOVER_STATUS_DISCOVER_RUNNING:
     {
 /* Multiple Card(s) detected - resolving */
 break;
             }
 case RF_DISCOVER_STATUS_DISCOVER_DONE:
     {
 /* Multiple Card(s) detected - resolved */
 /* Get card details to find the protocol/decide what data to send */
     err = RM_NFC_READER_PTX_CardRegistryGet(&g_rm_nfc_reader_ptx_instance,
&card_registry);
     assert(FSP_SUCCESS == err);
                        demo_state = IoTRd_DemoState_SelectCard;
 break;
             }
 default:
     {
 /* Unknown Discover-Status */
 break;
             }
             }
 break;
           }
 case IoTRd_DemoState_SelectCard:
     {
 /* Case for detection of multiple cards */
               err =
 RM_NFC_READER_PTX_CardActivate(&g_rm_nfc_reader_ptx_instance,
&card_registry->Cards[0], Prot_T2T);
     assert(FSP_SUCCESS == err);
                demo_state = IoTRd_DemoState_DataExchange;
```

```
 break;
            }
 case IoTRd_DemoState_DataExchange:
      {
 /* Case for data exchange due to detected card */
                err = RM_NFC_READER_PTX_DataExchange(&g_rm_nfc_reader_ptx_instance,
&nfc_send_buf);
      assert(FSP_SUCCESS == err);
                demo_state = IoTRd_DemoState_DeactivateReader;
 break;
            }
 case IoTRd_DemoState_DeactivateReader:
      {
                err =
 RM_NFC_READER_PTX_ReaderDeactivation(&g_rm_nfc_reader_ptx_instance,
NFC_READER_PTX_RETURN_DISCOVER);
      assert(FSP_SUCCESS == err);
                demo_state = IoTRd_DemoState_WaitForActivation;
 break;
            }
 case IoTRd_DemoState_SystemError:
      {
 /* System error */
 RM_NFC_READER_PTX_Close(&g_rm_nfc_reader_ptx_instance);
 while (1)
      {
 /* Do nothing */
              }
 break;
            }
 default:
      {
 break;
            }
```

```
        }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | nfc_reader_ptx_data_info_t |
| struct | nfc_reader_ptx_cfg_t |
| struct | nfc_reader_ptx_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | nfc_reader_ptx_ndef_read_write_t |
| enum | nfc_reader_ptx_power_mode_t |
| enum | nfc_reader_ptx_state_t |
| enum | nfc_reader_ptx_return_state_t |

## Data Structure Documentation

### ◆ nfc_reader_ptx_data_info_t

| struct nfc_reader_ptx_data_info_t | | |
|---|---|---|
| NFC data exchange packet parameters | | |
| Data Fields | | |
| uint8_t * | p_tx_buf | Pointer to buffer holding data to send. |
| uint32_t | tx_length | Length of data to send. |
| uint8_t * | p_rx_buf | Pointer to buffer for data to receive. |
| uint32_t | rx_length | Length of data to receive. |

### ◆ nfc_reader_ptx_cfg_t

| struct nfc_reader_ptx_cfg_t | | |
|---|---|---|
| NFC Configuration | | |
| Data Fields | | |
| bool | poll_type_a | Flag to indicate enabling discovery for Type-A tags. |
| bool | poll_type_b | Flag to indicate enabling discovery for Type-B tags. |

| | | |
|---|---|---|
| bool | poll_type_f | Flag to indicate enabling discovery for Type-F tags. |
| bool | poll_type_v | Flag to indicate enabling discovery for Type-V tags. |
| uint8_t | device_limit | Defines the maximum number of devices that can be connected at once. |
| uint8_t | discover_mode | Set polling mode, where 0 = Regular Polling, 1 = Low-Power Card Detection (LPCD), 2-255 = LPCD with every n-th Cycle Regular Polling. |
| uint32_t | idle_time_ms | RF-Discovery Loop. Idle time between polling cycles in milliseconds. |
| uint8_t | temp_sensor_calibrate | Flag to enable/disable temperature sensor calibration. |
| uint8_t | temp_sensor_shutdown | Expected thermal shutdown threshold value. |
| uint8_t | temp_sensor_ambient | Ambient temperature at which temperature sensor calibration takes place. |
| const bsp_io_port_pin_t | interrupt_pin | Interrupt pin for NFC chip to signal for communication. |
| ptxIoTRd_t * | iot_reader_context | Instance of the NFC SDK IoT Reader main structure. |
| rm_comms_instance_t * | p_comms_instance_ctrl | Pointer to the NFC module's Comms instance. |
| ioport_instance_t * | p_gpio_context | Pointer to the NFC module's GPIO instance. |
| external_irq_instance_t * | p_irq_context | Pointer to the NFC module's SPI external interrupt instance. |
| timer_instance_t * | p_timer_context | Pointer to the NFC module's sleep timer instance. |
| timer_instance_t * | p_app_timer | Pointer to the NFC module's Transparent Data Channel (TDC) timer instance. |
| ptxNDEF_t * | p_ndef_context | Pointer to the optional NDEF API context. |

### ◆ nfc_reader_ptx_instance_ctrl_t

| struct nfc_reader_ptx_instance_ctrl_t |
|---|
| NFC_READER_PTX private control block. DO NOT MODIFY. |

| Data Fields | | |
|---|---|---|
| uint8_t | ndef_work_buf[NFC_READER_PTX_NDEF_WORK_BUF_SIZE] | Buffer for NDEF internal data exchange. |
| uint32_t | open | Flag to indicate if NFC has been opened. |
| uint32_t | ndef_open | Flag to indicate if NFC NDEF has been initialized. |
| nfc_reader_ptx_state_t | state_flag | Flag to track Idle/discovered/activated. |
| nfc_reader_ptx_cfg_t const * | p_cfg | Pointer to p_cfg for NFC. |

## Enumeration Type Documentation

### ◆ nfc_reader_ptx_ndef_read_write_t

| enum nfc_reader_ptx_ndef_read_write_t | |
|---|---|
| NFC NDEF data exchange flags | |
| Enumerator | |
| NFC_READER_PTX_NDEF_WRITE | Write flag for NDEF data exchange. |
| NFC_READER_PTX_NDEF_READ | Read flag for NDEF data exchange. |

### ◆ nfc_reader_ptx_power_mode_t

| enum nfc_reader_ptx_power_mode_t | |
|---|---|
| NFC power mode flags | |
| Enumerator | |
| NFC_READER_PTX_ACTIVE_MODE | Active mode (default), all features enabled. |
| NFC_READER_PTX_STANDBY_MODE | Stand-by mode, most features disabled for low power. |

### ◆ nfc_reader_ptx_state_t

| enum nfc_reader_ptx_state_t | |
|---|---|
| NFC state machine status | |
| Enumerator | |
| NFC_READER_PTX_IDLE | Idle state before starting discovery. |
| NFC_READER_PTX_POLLING | Polling state for discovering tags. |
| NFC_READER_PTX_DISCOVERED | Discovered state for tags that were found. |
| NFC_READER_PTX_ACTIVATED | Activated state for activating a discovered tag. |

### ◆ nfc_reader_ptx_return_state_t

| enum nfc_reader_ptx_return_state_t | |
|---|---|
| NFC de-activation return state | |
| Enumerator | |
| NFC_READER_PTX_RETURN_IDLE | Rf-Deactivation Type 0 - Turn off RF-field. |
| NFC_READER_PTX_RETURN_DISCOVER | Rf-Deactivation Type 1 - Restart RF-Discovery. |
| NFC_READER_PTX_RETURN_SLEEP | Rf-Deactivation Type 2 - Put remote device to sleep. |
| NFC_READER_PTX_RETURN_SLEEP_NON_BLOCKING | Rf-Deactivation Type 3 - Put remote device to sleep (non-blocking) |
| NFC_READER_PTX_RETURN_NO_RF_RESET | Rf-Deactivation Type 4 - Common RF Deactivate but no Field Reset. |
| NFC_READER_PTX_RETURN_IDLE_PROTOCOL | Rf-Deactivation Type 5 - Use protocol-specific Deactivation, IDLE afterwards. |
| NFC_READER_PTX_RETURN_DISCOVER_PROTOCOL | Rf-Deactivation Type 6 - Use protocol-specific Deactivation, RF-Discovery started afterwards. |

## Function Documentation

### ◆ RM_NFC_READER_PTX_Open()

fsp_err_t RM_NFC_READER_PTX_Open ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, nfc_reader_ptx_cfg_t const *const *p_cfg* )

Initializes the NFC IoT Reader.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | p_cfg | Pointer to NFC IoT Reader configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_ALREADY_OPEN | Module is already opened. |
| FSP_ERR_INVALID_DATA | NFC SDK initialization data was incomplete. |

### ◆ RM_NFC_READER_PTX_DiscoveryStart()

fsp_err_t RM_NFC_READER_PTX_DiscoveryStart ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*)

Initiates discovery of NFC tags.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the idle state. |
| FSP_ERR_INVALID_DATA | NFC Discovery parameters were invalid or discovery was not started. |

### ◆ RM_NFC_READER_PTX_StatusGet()

fsp_err_t RM_NFC_READER_PTX_StatusGet ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, ptxIoTRd_StatusType_t *status_type*, uint8_t *const *p_status* )

Gets the status information on the provided status identifier (system, discovery, etc.). Used during polling to confirm a card was discovered.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | status_type | Status identifier for target info. |
| [in] | p_status | Pointer to location to store NFC chip status flag. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the polling state. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameters to NFC Status function. |

### ◆ RM_NFC_READER_PTX_CardRegistryGet()

fsp_err_t RM_NFC_READER_PTX_CardRegistryGet ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, ptxIoTRd_CardRegistry_t ** *pp_card_registry* )

Gets the internal card registry.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | pp_card_registry | Pointer to card registry for discovered cards. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameters to NFC registry function. |

### ◆ RM_NFC_READER_PTX_CardActivate()

fsp_err_t RM_NFC_READER_PTX_CardActivate ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, ptxIoTRd_CardParams_t *const *p_card_params*, ptxIoTRd_CardProtocol_t *protocol* )

Gets the details of the discovered card and connects to it. Not required if only a single tag was discovered.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | p_card_params | Parameters of the card to be activated. |
| [in] | protocol | The NFC protocol of the card to be activated. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the discovered/selection state. |

### ◆ RM_NFC_READER_PTX_DataExchange()

fsp_err_t RM_NFC_READER_PTX_DataExchange ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, nfc_reader_ptx_data_info_t *const *p_data_info* )

Sends and receives data to/from the activated card.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | p_data_info | Pointer to the NFC TX/RX data. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the activated state. |
| FSP_ERR_INVALID_ARGUMENT | Issue with input parameters (buffers) leading to failed transmission. |

### ◆ RM_NFC_READER_PTX_ReaderDeactivation()

fsp_err_t RM_NFC_READER_PTX_ReaderDeactivation ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, nfc_reader_ptx_return_state_t *return_state* )

Deactivates the activated card to return to the desired state (idle, discovery, etc.).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | return_state | Expectation for end state of NFC state machine. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the activated state. |
| FSP_ERR_INVALID_ARGUMENT | Issue with input parameters or transmission. |

### ◆ RM_NFC_READER_PTX_PowerModeSet()

fsp_err_t RM_NFC_READER_PTX_PowerModeSet ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, nfc_reader_ptx_power_mode_t *power_mode* )

Puts the device into either Active (all features) or Stand-by mode (most features off), to cut down energy consumption. RF communication is not possible from Stand-by mode, and Wake-up is possible only via communication interface. This function is only available exclusively before starting discovery.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | power_mode | Flag to indicate if setting to Active or Stand-by mode. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the idle/polling state. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input parameters to NFC Status function. |

### ◆ RM_NFC_READER_PTX_NDEF_Init()

fsp_err_t RM_NFC_READER_PTX_NDEF_Init ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, nfc_reader_ptx_data_info_t *const *p_data_info* )

Initializes the optional NDEF API add-on. Must be done after NFC stack is initialized.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | p_data_info | Pointer to the NFC TX/RX data. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_ALREADY_OPEN | NDEF API add-on was already initialized. |
| FSP_ERR_UNSUPPORTED | NDEF API add-on was not included in project configuration. |

### ◆ RM_NFC_READER_PTX_NDEF_DataExchange()

fsp_err_t RM_NFC_READER_PTX_NDEF_DataExchange ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*, uint8_t * *p_ndef_message*, uint32_t *message_length*, nfc_reader_ptx_ndef_read_write_t *ndef_read_write_flag* )

Exchanges NDEF data with an NFC tag by reading from or writing to it.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |
| [in] | p_ndef_message | Pointer to the buffer to transmit or receive a message. |
| [in] | message_length | Length of the NDEF message buffer. |
| [in] | ndef_read_write_flag | Flag to control if the data exchange is a read or a write. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_NOT_INITIALIZED | NDEF add-on is not initialized yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the activated state. |
| FSP_ERR_INVALID_DATA | Problem with input parameters or data transmission. |
| FSP_ERR_UNSUPPORTED | NDEF API add-on was not included in project configuration. |

### ◆ RM_NFC_READER_PTX_NDEF_Lock()

fsp_err_t RM_NFC_READER_PTX_NDEF_Lock ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*)

Locks an NFC tag to prevent writing to it, making it read-only.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened yet. |
| FSP_ERR_NOT_INITIALIZED | NDEF add-on is not initialized yet. |
| FSP_ERR_INVALID_STATE | NFC module is not in the activated state. |
| FSP_ERR_INVALID_DATA | Problem with input parameters or data transmission. |
| FSP_ERR_UNSUPPORTED | NDEF API add-on was not included in project configuration. |

### ◆ RM_NFC_READER_PTX_Close()

fsp_err_t RM_NFC_READER_PTX_Close ( nfc_reader_ptx_instance_ctrl_t *const *p_ctrl*)

Closes the FSP NFC module and resets all variables.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to NFC IoT Reader control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Assertion error occurred. |
| FSP_ERR_NOT_OPEN | Close being called before the module is Open. |

## 5.2.12.25 NetX Duo Ethernet Driver (rm_netxduo_ether)

Modules » Networking

# Overview

This module provides a NetX Duo driver that is implemented using the Ethernet Interface.

Please refer to the NetXDuo documentation for further details.

### Features

- Packet Types Supported
  - ARP
  - IPv4
  - IPv6
- Link status callback
- Configurable IP MTU

# Configuration

### Configurations for Networking > NetX Duo Ethernet Driver (rm_netxduo_ether)

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_netxduo_ether_0 | Module name. |
| IP MTU | Value must be in the range [576, 1500] bytes. | 1500 | IP MTU |

# Usage Notes

### Calculating the Packet Size for an IP instance

In order to ensure that there is enough space to store an entire Ethernet frame, the packet pool used for receiving packets must have a payload size that is 32 bytes larger than the configured ether_cfg_t::ether_buffer_size. The extra 32 bytes is needed in order to ensure that the allocated packets are properly aligned to 32 bytes.

ether_cfg_t::ether_buffer_size is calculated from the IP MTU using the following formula:

ceil(( rm_netxduo_ether_cfg_t::mtu + Ethernet Header (14) + Padding Bytes (2)) / 32 ) * 32

### Zero Copy Support

Zero copy support should only be disabled when a lower level ethernet driver doesn't support it. Disabling zero copy will negatively impact performance. When zero copy is disabled larger buffers will need to be allocated or link speed will need to be limited to counteract poor performance.

# Examples

### Basic Example

This is a basic example of minimal use of the NetX Duo Ether Driver in an application.

```
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)
#define NETXDUO_EXAMPLE_ARP_CACHE_SIZE (2048U)
#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)
#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
#define NETXDUO_EXAMPLE_MAC_ADDRESS_MSW (0)
#define NETXDUO_EXAMPLE_MAC_ADDRESS_LSW (0)
static NX_IP g_ip;
static NX_PACKET_POOL g_packet_pool;
static uint8_t g_ip_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_ip_arp_cache_memory[NETXDUO_EXAMPLE_ARP_CACHE_SIZE]
BSP_ALIGN_VARIABLE(4);
static void rm_netxduo_ether0 (NX_IP_DRIVER * driver_req_ptr)
{
 /* Pass the driver request and ethernet driver configuration to the NetX Duo Ether
Driver. */
    rm_netxduo_ether(driver_req_ptr, &g_netxduo_ether_instance);
}
void rm_netxduo_ether_example ()
{
    UINT status;
    nx_system_initialize();
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool,
 "Packet Pool",
                                   NETXDUO_EXAMPLE_PACKET_SIZE,
                                   &g_packet_pool_memory[0],
```

```
                                    NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_ether driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip,
 "IP Instance",
                          IP_ADDRESS(192, 168, 1, 2),
                          IP_ADDRESS(255, 255, 255, 0),
                          &g_packet_pool,
                          rm_netxduo_ether0,
                          &g_ip_stack_memory[0],
 sizeof(g_ip_stack_memory),
                          0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_arp_enable(&g_ip, g_ip_arp_cache_memory, sizeof
(g_ip_arp_cache_memory));
    assert(NX_SUCCESS == status);
    status = nx_tcp_enable(&g_ip);
    assert(NX_SUCCESS == status);
    status = nx_icmp_enable(&g_ip);
    assert(NX_SUCCESS == status);
}
```

### Changing MAC address Example

This is an example of changing the MAC address after an IP instance has already been created.

```
void rm_netxduo_ether_change_mac_address_example ()
{
    ULONG driver_status;
    UINT status;
 /* Disable the link. */
    status = nx_ip_driver_interface_direct_command(&g_ip, NX_LINK_DISABLE, 0,
&driver_status);
```

```
    assert(NX_SUCCESS == status);
 /* Update the MAC address. */
    status = nx_ip_interface_physical_address_set(&g_ip,
                                                  0,
                                                  NETXDUO_EXAMPLE_MAC_ADDRESS_MSW,
                                                  NETXDUO_EXAMPLE_MAC_ADDRESS_LSW,
                                                  NX_TRUE);
    assert(NX_SUCCESS == status);
 /* Re-enable the link. */
    status = nx_ip_driver_interface_direct_command(&g_ip, NX_LINK_ENABLE, 0,
&driver_status);
    assert(NX_SUCCESS == status);
}
```

#### 5.2.12.26 NetX Duo WiFi Driver (rm_netxduo_wifi)

Modules » Networking

# Overview

This module provides a NetX Duo driver that is implemented using the rm_wifi_onchip_silex driver.

Please refer to the NetXDuo documentation for further details.

**Features**

- Packet Types Supported
  - TCP/IPv4
  - UDP
- Configurable IP MTU

# Configuration

**Build Time Configurations for rm_netxduo_wifi**

The following build time configurations are defined in fsp_cfg/middleware/rm_netxduo_wifi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| IP MTU (bytes) | Value must be in the | 1500 | IP MTU |

range [576, 1500] bytes.

# Usage Notes

### Connecting to a Wireless Network

NetX Duo does not support connecting to a wireless network directly and instead the driver functions should be called directly to establish a connection.

- rm_wifi_onchip_silex_open() should be called to initialize the WiFi module driver.
- rm_wifi_onchip_silex_scan() should be used to scan for access points.
- rm_wifi_onchip_silex_connect() should be used before opening a NetX IP instance in order to connect it to a network.
- rm_wifi_onchip_silex_network_info_get() should be used after an access point connection has been established to get DHCP acquired IP address, subnet mask, and gateway.

### Unsupported NetX Duo Features

These features are currently not supported due to lack of support in the rm_wifi_onchip_silex driver or in the WiFi module hardware/firmware:

- IPv6 functionality
- TCP/UDP server mode
- ARP is handled directly by the WiFi module
- DHCP is always enabled on module by the rm_wifi_onchip_silex driver, thus the software DHCP stack is unsupported.
- The source port will not be used when binding a TCP or UDP socket. Applications should set the source port to NX_ANY_PORT.
- Use of the NetX Duo FTP addon is not recommended as certain FTP functions will return a NX_NO_PACKET due to how SX-ULPGN firmware handles bundling packet data.

### On Module Alternatives

Instead of using NetX Duo Software Addons the following features can be performed directly on the WiFi module by calling the appropriate function instead if desired:

- DNS: The user can preform a DNS lookup by calling rm_wifi_onchip_silex_dns_query().
- ICMP: The user can perform a ping by calling rm_wifi_onchip_silex_ping().
- SNTP: The WiFi module has multiple public APIs for time related functions:
  - RM_WIFI_ONCHIP_SILEX_EpochTimeGet()
  - RM_WIFI_ONCHIP_SILEX_LocalTimeGet()
  - RM_WIFI_ONCHIP_SILEX_SntpEnableSet()
  - RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet()
  - RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet()

# Examples

### Basic Example

This is a basic example of minimal use of the NetX Duo WiFi Driver in an application.

```
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)
```

```
#define NETXDUO_EXAMPLE_ARP_CACHE_SIZE (2048U)
#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)
#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
#define NETXDUO_EXAMPLE_SSID "ssidName"
#define NETXDUO_EXAMPLE_PASSWORD "password"
static NX_IP g_ip0;
static NX_PACKET_POOL g_packet_pool0;
static uint8_t g_ip0_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool0_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
extern wifi_onchip_silex_cfg_t g_wifi_onchip_silex_cfg;
void rm_netxduo_wifi_example ()
{
    UINT      status;
 fsp_err_t err;
    nx_system_initialize();
 /* Open WiFi module */
    err = rm_wifi_onchip_silex_open(&g_wifi_onchip_silex_cfg);
    assert(FSP_SUCCESS == err);
 /* Connect to desired AP */
    err = rm_wifi_onchip_silex_connect(NETXDUO_EXAMPLE_SSID, eWiFiSecurityWPA2,
NETXDUO_EXAMPLE_PASSWORD);
    assert(FSP_SUCCESS == err);
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool0,
 "Packet Pool",
                                   NETXDUO_EXAMPLE_PACKET_SIZE,
                                   &g_packet_pool0_pool_memory[0],
                                   NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
```

```
 /* Create an IP instance using the rm_netxduo_wifi driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip0,
 "IP Instance",
                          IP_ADDRESS(192, 168, 1, 2),
                          IP_ADDRESS(255, 255, 255, 0),
                          &g_packet_pool0,
                          rm_netxduo_wifi,
                          &g_ip0_stack_memory[0],
 sizeof(g_ip0_stack_memory),
                          0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_tcp_enable(&g_ip0);
    assert(NX_SUCCESS == status);
    status = nx_udp_enable(&g_ip0);
    assert(NX_SUCCESS == status);
}
```

### TLS Example

This is a basic example of connecting a TLS socket.

```
#define NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE (18000U)
#define NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE (4000U)
#define NETXDUO_EXAMPLE_PORT (3005U)
#define NETXDUO_EXAMPLE_SERVER_PORT (9050U)
#define NETXDUO_EXAMPLE_IP IP_ADDRESS(1, 2, 3, 5)
NX_SECURE_TLS_CRYPTO       g_nx_crypto_tls_test_ciphers;
static UCHAR g_tls_crypto_metadata[NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE];
static UCHAR g_tls_packet_buffer[NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE];
static NX_SECURE_X509_CERT g_certificate;
extern const UCHAR g_trusted_ca_data; // User trusted certificates
extern USHORT g_trusted_ca_length;
void rm_netxduo_wifi_tls_example ()
```

```
{
    UINT                  status;
 fsp_err_t             err;
    NX_TCP_SOCKET       * p_socket;
    NX_SECURE_TLS_SESSION tls_session;
    nx_system_initialize();
 /* Initialize NetX Crypto */
    status = nx_crypto_initialize();
    assert(NX_SUCCESS == status);
 /* Open WiFi module */
    err = rm_wifi_onchip_silex_open(&g_wifi_onchip_silex_cfg);
    assert(FSP_SUCCESS == err);
 /* Connect to desired AP */
    err = rm_wifi_onchip_silex_connect(NETXDUO_EXAMPLE_SSID, eWiFiSecurityWPA2,
NETXDUO_EXAMPLE_PASSWORD);
    assert(FSP_SUCCESS == err);
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool0,
 "Packet Pool",
                                   NETXDUO_EXAMPLE_PACKET_SIZE,
                                   &g_packet_pool0_pool_memory[0],
                                   NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_wifi driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip0,
 "IP Instance",
                          IP_ADDRESS(192, 168, 1, 2),
                          IP_ADDRESS(255, 255, 255, 0),
                          &g_packet_pool0,
                          rm_netxduo_wifi,
                          &g_ip0_stack_memory[0],
 sizeof(g_ip0_stack_memory),
                          0);
```

```
   assert(NX_SUCCESS == status);
/* Enable all modules that are required by the application. */
   status = nx_tcp_enable(&g_ip0);
   assert(NX_SUCCESS == status);
/* Initialize the NetX Secure TLS system. */
   nx_secure_tls_initialize();
/* Create a TCP socket to use for the TLS session. */
   status = nx_tcp_socket_create(&g_ip0,
                                 p_socket,
"TLS Client Socket",
                                 NX_IP_NORMAL,
                                 NX_FRAGMENT_OKAY,
                                 NX_IP_TIME_TO_LIVE,
                                 1024 * 4,
                                 NX_NULL,
                                 NX_NULL);
   assert(NX_SUCCESS == status);
/* Create a TLS session for our socket. This sets up the TLS session object for
 * later use */
   status =
      nx_secure_tls_session_create(&tls_session,
                                   &g_nx_crypto_tls_test_ciphers,
                                   g_tls_crypto_metadata,
                                   NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE);
   assert(NX_SUCCESS == status);
/* Set the packet reassembly buffer for this TLS session. */
   status = nx_secure_tls_session_packet_buffer_set(&tls_session,
                                                    g_tls_packet_buffer,

NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE);
   assert(NX_SUCCESS == status);
/* Initialize an X.509 certificate with the CA root certificate data. */
   status = nx_secure_x509_certificate_initialize(&g_certificate,
                                                  (UCHAR *) g_trusted_ca_data,
```

```
                                           g_trusted_ca_length,
                                NX_NULL,
                                           0,
                                NX_NULL,
                                           0,
                                           NX_SECURE_X509_KEY_TYPE_NONE);
    assert(NX_SUCCESS == status);
 /* Add the initialized certificate as a trusted root certificate. */
    status = nx_secure_tls_trusted_certificate_add(&tls_session, &g_certificate);
    assert(NX_SUCCESS == status);
 /* Bind the socket to a port. */
    status = nx_tcp_client_socket_bind(p_socket, NETXDUO_EXAMPLE_PORT,
NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
 /* Connect TCP socket */
    status = nx_tcp_client_socket_connect(p_socket, NETXDUO_EXAMPLE_IP,
NETXDUO_EXAMPLE_SERVER_PORT, NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
 /* Start the TLS Session using the connected TCP socket. This function will
  * ascertain from the TCP socket state that this is a TLS Client session. */
    status = nx_secure_tls_session_start(&tls_session, p_socket, NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
}
```

### UDP Example

This is a basic example of connecting a UDP socket.

```
#define NETXDUO_TESTS_SOCKET_TIMEOUT (3000)
#define NETXDUO_TESTS_UDP_TX_PORT (5000)
#define NETXDUO_TESTS_SERVER_IP (0xC0A80064)
void rm_netxduo_wifi_udp_example (void)
{
    NX_UDP_SOCKET udp_client_socket;
 /* IP Address of the remote socket. */
```

```
    NXD_ADDRESS g_ip_address = NETXDUO_TESTS_SERVER_IP;
 /* Create the socket. */
    nx_udp_socket_create(&g_ip, &udp_client_socket, "Socket 0", NX_IP_NORMAL,
NX_FRAGMENT_OKAY, 0, 5);
 /* Bind to the UDP port. */
    nx_udp_socket_bind(&udp_client_socket, NETXDUO_TESTS_UDP_TX_PORT,
TX_WAIT_FOREVER);
    NX_PACKET * p_nx_packet = NX_NULL;
 /* Allocate a packet for message data */
    nx_packet_allocate(&g_packet_pool0, &p_nx_packet, NX_UDP_PACKET,
NETXDUO_TESTS_SOCKET_TIMEOUT)
 /* Append message data to packet */
    nx_packet_data_append(&p_nx_packet, "Hello World", 12, &g_packet_pool0,
NX_WAIT_FOREVER);
 /* Send UDP message to server */
    nx_udp_socket_send(&udp_client_socket, &p_nx_packet, &g_ip_address,
NETXDUO_TESTS_UDP_TX_PORT);
    NX_PACKET * p_nx_packet_recv;
 /* Receive any response from UDP server */
    nx_udp_socket_receive(&udp_client_socket, &p_nx_packet_recv,
NETXDUO_TESTS_SOCKET_TIMEOUT);
 /* Unbind and delete the udp socket. */
    nx_udp_socket_unbind(&udp_client_socket);
    nx_udp_socket_delete(&udp_client_socket);
}
```

### 5.2.12.27 On Chip HTTP Client on DA16XXX (rm_http_onchip_da16xxx)

Modules » Networking

#### Functions

| | |
|---|---|
| fsp_err_t | RM_HTTP_DA16XXX_Open (http_onchip_da16xxx_instance_ctrl_t *p_ctrl, http_onchip_da16xxx_cfg_t const *const p_cfg) |
| fsp_err_t | RM_HTTP_DA16XXX_Send (http_onchip_da16xxx_instance_ctrl_t *p_ctrl, http_onchip_da16xxx_request_t *p_request, |

| | |
|---|---|
| | http_onchip_da16xxx_buffer_t *p_buffer) |
| fsp_err_t | RM_HTTP_DA16XXX_Start (http_onchip_da16xxx_instance_ctrl_t *p_ctrl, http_onchip_da16xxx_ssl_enable_t enable) |
| fsp_err_t | RM_HTTP_DA16XXX_Stop (http_onchip_da16xxx_instance_ctrl_t *p_ctrl) |
| fsp_err_t | RM_HTTP_DA16XXX_Close (http_onchip_da16xxx_instance_ctrl_t *p_ctrl) |

### Detailed Description

HTTP client implementation using the DA16XXX WiFi module on RA MCUs.

# Overview

This Middleware module supplies an implementation for the HTTP Client on the DA16XXX module, as well as the ability to start an HTTP server on the module. The network stack is run on the DA16XXX and this module inherits and extends the core functionality from the DA16XXX WiFi driver. The DA16XXX HTTP module only supports FreeRTOS currently (Microsoft Azure support expected in future release).

The DA16XXX module supports the following modules:

- DA16200
- DA16600

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Features

The HTTP Onchip da16xxx Middleware driver supplies these features:

- Supports sending a request header (GET, PUT, and POST) to an HTTP server and receiving a response header
- Supports unsecure and secure connection via TLS encryption
- Supports parsing of the response header and returning to the user
- Supports other optional configurations such as Server Name Indication (SNI) and ALPNs
- Supports starting a secure or unsecure HTTP server on the DA16XXX module

# Configuration

### Build Time Configurations for rm_http_onchip_da16xxx

The following build time configurations are defined in fsp_cfg/rm_http_onchip_da16xxx_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Networking > HTTP Client/Server on DA16XXX (rm_http_onchip_da16xxx)

This module can be added to the Stacks tab via New Stack > Networking > HTTP Client/Server on DA16XXX (rm_http_onchip_da16xxx).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Certificates | | | |
| HTTP Certificates Header File | Must be a valid C symbol | | Name of header file that will contain certificates (macros). User must create header file if this field is populated. |
| Root CA | Must be a valid C symbol | ROOT_CA | Links to user-defined macro of the same name for Root CA which user must define in application header. |
| Client Certificate | Must be a valid C symbol | CLIENT_CERT | Links to user-defined macro of the same name for client certificate which user must define in application header. |
| Private Key | Must be a valid C symbol | PRIVATE_KEY | Links to user-defined macro of the same name for private key which user must define in application header. |
| ALPN | | | |
| ALPN 1 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| ALPN 2 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| ALPN 3 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| Server Name Indication (SNI) | Manual Entry | | |
| TLS Authentification Level | • 0 (None)<br>• 1 (Optional)<br>• 2 (Required) | 0 (None) | Level of TLS authentication for HTTP requests. |

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Interrupt Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Clock Configuration

Refer to UART (r_sci_uart).

### Pin Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx)

## Usage Notes

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

- When the user intends to use secure TLS in their application, they need to provide certificates (root certificate authority, client certificate, and private key) in a header file
- The user must provide certificate macro names in the FSP Configurator properties, as well as the name of the header file (to link the module, certificates, and header together)
- If the user provides these credentials in the Configurator, they must create a new header file in their /src/ folder with the same name as they provided in the Configurator properties. When the code is generated, a build error will occur if the header file is not created
- When starting an HTTP server, the DA16XXX module will use the default HTTP test server included in the SDK. The user will need to customize the SDK to provide their own desired server. Refer to guides on the DA16XXX module page for importing/modifying the SDK

### Limitations

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

- This module is not a standalone module - ensure that the WiFi module APIs are invoked first to configure the DA16XXX module and connect to an Access Point before using the HTTP module
- Ensure that DA16XXX SDK version v3.2.6 or later is being used, where the default buffer size for incoming TLS is 6KB and outgoing TLS is 4KB. Using earlier versions may result in failure to connect to an HTTP server with TLS (unless buffer size settings are altered manually)

## Examples

### Basic Example

This is a basic example of minimal use of HTTP Middleware in an application.

```
void http_onchip_basic_example (void)
{
 WIFIReturnCode_t wifi_err;
 fsp_err_t       http_err;
```

```
/* Setup Access Point connection parameters */
WIFINetworkParams_t net_params =
   {
      .ucChannel                  = 0,
      .xPassword.xWPA.cPassphrase = WIFI_PWD,
      .ucSSID                     = WIFI_SSID,
      .xPassword.xWPA.ucLength = sizeof(WIFI_PWD),
      .ucSSIDLength            = sizeof(WIFI_SSID),
      .xSecurity               = eWiFiSecurityWPA2,
   };
/* Open connection to the Wifi Module */
   wifi_err = WIFI_On();
   assert(eWiFiSuccess == wifi_err);
/* Connect to the Access Point */
   wifi_err = WIFI_ConnectAP(&net_params);
   assert(eWiFiSuccess == wifi_err);
/* Initialize the HTTP Client connection */
   http_err = RM_HTTP_DA16XXX_Open(&g_rm_http_onchip_da16xxx_instance,
&g_http_onchip_da16xxx_cfg);
   assert(FSP_SUCCESS == http_err);
/* Initialize the HTTP Client connection */
   http_err = RM_HTTP_DA16XXX_Send(&g_rm_http_onchip_da16xxx_instance, &test_req,
&user_buf);
   assert(FSP_SUCCESS == http_err);
/* Close the HTTP Client module */
   http_err = RM_HTTP_DA16XXX_Close(&g_rm_http_onchip_da16xxx_instance);
   assert(FSP_SUCCESS == http_err);
}
```

## Data Structures

| | |
|---|---|
| struct | http_onchip_da16xxx_request_t |
| struct | http_onchip_da16xxx_buffer_t |
| struct | http_onchip_da16xxx_cfg_t |

| | |
|---|---|
| struct | http_onchip_da16xxx_instance_ctrl_t |

### Macros

| | |
|---|---|
| #define | HTTP_ONCHIP_DA16XXX_MAX_ALPN<br>Maximum number of ALPNs supported by DA16XXX. |
| #define | HTTP_ONCHIP_DA16XXX_MAX_SNI_LEN<br>Maximum length of SNI supported by DA16XXX. |
| #define | HTTP_ONCHIP_DA16XXX_MAX_GET_LEN<br>Maximum length of HTTP GET request. |

### Enumerations

| | |
|---|---|
| enum | http_onchip_da16xxx_method_t |
| enum | http_onchip_da16xxx_tls_auth_t |
| enum | http_onchip_da16xxx_ssl_enable_t |

## Data Structure Documentation

### ◆ http_onchip_da16xxx_request_t

| struct http_onchip_da16xxx_request_t | | |
|---|---|---|
| HTTP Request Buffer | | |
| Data Fields | | |
| const char * | p_http_endpoint | Defines the URL to send HTTP requests to. |
| const char * | p_request_body | Pointer to optional buffer for body. |
| http_onchip_da16xxx_method_t | method | Request method to be used. |
| uint32_t | length | Length of optional body buffer. |

### ◆ http_onchip_da16xxx_buffer_t

| struct http_onchip_da16xxx_buffer_t | | |
|---|---|---|
| HTTP User Buffers | | |
| Data Fields | | |
| char * | p_request_buffer | User HTTP request buffer. |
| uint32_t | req_length | Size of HTTP request buffer. |

| | | |
|---|---|---|
| char * | p_response_buffer | User HTTP request buffer. |
| uint32_t | resp_length | Size of HTTP request buffer. |

### ◆ http_onchip_da16xxx_cfg_t

| struct http_onchip_da16xxx_cfg_t | | |
|---|---|---|
| HTTP Configuration | | |
| Data Fields | | |
| at_transport_da16xxx_instance_t const * | p_transport_instance | |
| const char * | p_alpns[ HTTP_ONCHIP_DA16XXX_MAX_ALPN] | Buffer for storing ALPN names. |
| uint8_t | alpn_count | ALPN Protocols count. Max value is 3. |
| const char * | p_sni_name | Pointer to Server Name Indication. |
| http_onchip_da16xxx_tls_auth_t | tls_level | Set TLS Authentication Level (0, 1, or 2) |
| const char * | p_root_ca | String representing a trusted server root certificate. |
| uint32_t | root_ca_size | Size associated with root CA Certificate. |
| const char * | p_client_cert | String representing a Client certificate. |
| uint32_t | client_cert_size | Size associated with Client certificate. |
| const char * | p_client_private_key | String representing Client Private Key. |
| uint32_t | private_key_size | Size associated with Client Private Key. |

### ◆ http_onchip_da16xxx_instance_ctrl_t

| struct http_onchip_da16xxx_instance_ctrl_t | | |
|---|---|---|
| HTTP_ONCHIP_DA16XXX private control block. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | open | Flag to indicate if HTTP has been opened. |
| http_onchip_da16xxx_ssl_enable_t | enable | Flag to indicate if HTTP server is secure. |
| http_onchip_da16xxx_cfg_t | p_cfg | Pointer to p_cfg for HTTP. |

| | | |
|---|---|---|
| const * | | |

## Enumeration Type Documentation

### ◆ http_onchip_da16xxx_method_t

| enum http_onchip_da16xxx_method_t | |
|---|---|
| HTTP Request Method | |
| Enumerator | |
| HTTP_ONCHIP_DA16XXX_GET | Value converted to "get" string. |
| HTTP_ONCHIP_DA16XXX_POST | Value converted to "post" string. |
| HTTP_ONCHIP_DA16XXX_PUT | Value converted to "put" string. |

### ◆ http_onchip_da16xxx_tls_auth_t

| enum http_onchip_da16xxx_tls_auth_t | |
|---|---|
| HTTP TLS Authentication Level | |
| Enumerator | |
| HTTP_ONCHIP_DA16XXX_TLS_VERIFY_NONE | TLS verification is not used. |
| HTTP_ONCHIP_DA16XXX_TLS_VERIFY_OPTIONAL | TLS verification is optional. |
| HTTP_ONCHIP_DA16XXX_TLS_VERIFY_REQUIRED | TLS verification is required. |

### ◆ http_onchip_da16xxx_ssl_enable_t

| enum http_onchip_da16xxx_ssl_enable_t | |
|---|---|
| HTTP SSL Enable | |
| Enumerator | |
| HTTP_ONCHIP_DA16XXX_SSL_DISABLE | SSL is disabled. |
| HTTP_ONCHIP_DA16XXX_SSL_ENABLE | SSL is enabled. |

## Function Documentation

### ◆ RM_HTTP_DA16XXX_Open()

fsp_err_t RM_HTTP_DA16XXX_Open ( http_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, http_onchip_da16xxx_cfg_t const *const *p_cfg* )

Initialize the DA16XXX on-chip HTTP Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to HTTP Client instance control structure. |
| [in] | p_cfg | Pointer to HTTP Client configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Parameter checking was unsuccessful. |
| FSP_ERR_INVALID_ARGUMENT | Data size is too large or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |

### ◆ RM_HTTP_DA16XXX_Send()

fsp_err_t RM_HTTP_DA16XXX_Send ( http_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, http_onchip_da16xxx_request_t * *p_request*, http_onchip_da16xxx_buffer_t * *p_buffer* )

Send the HTTP request with the configured buffers.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to HTTP Client instance control structure. |
| [in] | p_request | Pointer to HTTP request control structure. |
| [in] | p_buffer | Pointer to HTTP user buffer struct for request and response. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Parameter checking was unsuccessful. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Data size is too large. |
| FSP_ERR_INVALID_DATA | Data was not received correctly. |

### ◆ RM_HTTP_DA16XXX_Start()

fsp_err_t RM_HTTP_DA16XXX_Start ( http_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, http_onchip_da16xxx_ssl_enable_t *enable* )

Start the HTTP server.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to HTTP Client instance control structure. |
| [in] | enable | Enum value to enable or disable SSL. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Parameter checking was unsuccessful. |
| FSP_ERR_INVALID_ARGUMENT | SSl enable is either null or incorrect. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ RM_HTTP_DA16XXX_Stop()

fsp_err_t RM_HTTP_DA16XXX_Stop ( http_onchip_da16xxx_instance_ctrl_t * *p_ctrl*)

Start the HTTP server.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to HTTP Client instance control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | Parameter checking was unsuccessful. |
| FSP_ERR_INVALID_ARGUMENT | SSl enable is either null or incorrect. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ RM_HTTP_DA16XXX_Close()

fsp_err_t RM_HTTP_DA16XXX_Close ( http_onchip_da16xxx_instance_ctrl_t * *p_ctrl*)

Close the DA16XXX HTTP Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to HTTP Client instance control structure. |

**Return values**

| | |
|---|---|
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | Parameter checking was unsuccessful. |

## 5.2.12.28 On Chip MQTT Client on DA16XXX (rm_mqtt_onchip_da16xxx)

Modules » Networking

**Functions**

| | |
|---|---|
| fsp_err_t | RM_MQTT_DA16XXX_Open (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl, mqtt_onchip_da16xxx_cfg_t const *const p_cfg) |
| fsp_err_t | RM_MQTT_DA16XXX_Disconnect (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl) |
| fsp_err_t | RM_MQTT_DA16XXX_Connect (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl, uint32_t timeout_ms) |
| fsp_err_t | RM_MQTT_DA16XXX_Publish (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl, mqtt_onchip_da16xxx_pub_info_t *const p_pub_info) |
| fsp_err_t | RM_MQTT_DA16XXX_Subscribe (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl, mqtt_onchip_da16xxx_sub_info_t *const p_sub_info, size_t subscription_count) |
| fsp_err_t | RM_MQTT_DA16XXX_UnSubscribe (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl, mqtt_onchip_da16xxx_sub_info_t *const p_sub_info) |
| fsp_err_t | RM_MQTT_DA16XXX_Receive (mqtt_onchip_da16xxx_instance_ctrl_t *p_ctrl) |
| fsp_err_t | RM_MQTT_DA16XXX_Close (mqtt_onchip_da16xxx_instance_ctrl_t |

| | |
|---|---|
| | *p_ctrl) |

### Detailed Description

MQTT client implementation using the DA16XXX WiFi module on RA MCUs.

# Overview

This Middleware module supplies an implementation for the MQTT Client on the DA16XXX module. The network stack is run on the DA16XXX and this module inherits and extends the core functionality from the DA16XXX WiFi driver. The DA16XXX MQTT module only supports FreeRTOS currently (Microsoft Azure support expected in future release).

The DA16XXX module supports the following modules:

- DA16200
- DA16600

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Features

The MQTT Onchip da16xxx Middleware driver supplies these features:

- Supports connect/disconnect to an MQTT broker via hostname, port, and user credentials
- Supports unsecure and secure connection via TLS encryption
- Supports the MQTT subscribe/publish model for multiple topics
- Supports other optional configurations such as MQTT v3.1.1, Quality-of-service (QoS) level, TLS cipher suites, and ALPNs

# Configuration

### Build Time Configurations for rm_mqtt_onchip_da16xxx

The following build time configurations are defined in fsp_cfg/rm_mqtt_onchip_da16xxx_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Size of MQTT RX buffer | Must be an integer greater than 0 | 512 | Size in bytes of the MQTT buffer used for receiving subscribed data. Must be an integer greater than 0. |
| Size of MQTT TX buffer | Must be an integer greater than 200 and less than 2064 | 512 | Size in bytes of the MQTT buffer used for sending commands and publishing data. Maximum publishing |

length is 2063 bytes

### Configurations for Networking > MQTT Client on DA16XXX (rm_mqtt_onchip_da16xxx)

This module can be added to the Stacks tab via New Stack > Networking > MQTT Client on DA16XXX (rm_mqtt_onchip_da16xxx).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Certificates | | | |
| MQTT Certificates Header File | Must be a valid C symbol | | Name of header file that will contain certificates (macros). User must create header file if this field is populated. |
| Root CA | Must be a valid C symbol | ROOT_CA | Links to user-defined macro of the same name for Root CA which user must define in application header. |
| Client Certificate | Must be a valid C symbol | CLIENT_CERT | Links to user-defined macro of the same name for client certificate which user must define in application header. |
| Private Key | Must be a valid C symbol | PRIVATE_KEY | Links to user-defined macro of the same name for private key which user must define in application header. |
| ALPN | | | |
| ALPN 1 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| ALPN 2 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| ALPN 3 | Manual Entry | | Select Application Layer Protocol Negotiations (ALPNs) |
| Interrupts | | | |
| Callback | Must be a valid C symbol | mqtt0_callback | A user callback function. This callback function must be provided. It is called from the MQTT Receive function to process |

| | | | |
|---|---|---|---|
| | | | subscribed MQTT messages. |
| Use MQTT protocol v3.1.1 | • Disabled<br>• Enabled | Enabled | |
| MQTT Receive Maximum Timeout (ms) | Timeout must be an integer greater than 0 and less than 65535 | 10 | Timeout for the MQTT Receive function to check the buffer for incoming MQTT messages in milliseconds. |
| MQTT Transit Maximum Timeout (ms) | Timeout must be an integer greater than 0 and less than 65535 | 10 | Timeout for publishing MQTT messages in milliseconds. |
| Clean Session | • Disabled<br>• Enabled | Enabled | |
| Keep Alive (s) | Must be an integer greater than 0 and less than 65535 | 60 | MQTT ping period to check if connection is still active. |
| Client Identifier | Manual Entry | | |
| Host Name (or IP address) | Manual Entry | | |
| MQTT Port | Must be an integer greater than 0 and less than 65535 | 8883 | |
| MQTT User Name | Must be a valid C symbol | | |
| MQTT Password | Must be a valid C symbol | | |
| Last Will Topic | Manual Entry | | Topic for MQTT Last Will message. |
| Last Will Message | Manual Entry | | Payload for MQTT Last Will message. |
| Server Name Indication (SNI) | Manual Entry | | |
| Last Will QoS | • 0<br>• 1<br>• 2 | 0 | Quality-of-Service for MQTT Last Will message. |
| TLS Cipher Suites | Refer to the RA Configuration tool for available options. | 0U | Select TLS Cipher Suites |

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

Note: This module provides the basic configurations in the FSP Configurator required to connect to an MQTT broker such as credential information (user, password, client key, etc.), publish/subscribe

topics, and optional configurations as required. The user can then copy these configurations to a new structure if they want to connect to a new endpoint/make changes to the configurations dynamically.

### Interrupt Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Clock Configuration

Refer to UART (r_sci_uart).

### Pin Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx)

## Usage Notes

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

- This module is not a standalone module - ensure that the WiFi module APIs are invoked first to configure the DA16XXX module and connect to an Access Point before using the MQTT module where the default buffer size for incoming TLS is 6KB and outgoing TLS is 4KB.
- When the user intends to use secure TLS in their application, they need to provide certificates (root certificate authority, client certificate, and private key) in a header file
- The user must provide certificate macro names in the FSP Configurator properties, as well as the name of the header file (to link the module, certificates, and header together)
- If the user provides these credentials in the Configurator, they must create a new header file in their /src/ folder with the same name as they provided in the Configurator properties. When the code is generated, a build error will occur if the header file is not created
- If only using username and password, these fields can be left blank (default), and a header file does not need to be created

### Limitations

Refer to WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

- Ensure that DA16XXX SDK version v3.2.6 or later is being used, where the default buffer size for incoming TLS is 6KB and outgoing TLS is 4KB. Using earlier versions may result in failure to connect to an MQTT broker with TLS (unless buffer size settings are altered manually)
- For publish/subscribe topics, a separate QoS level cannot be set per topic - only one QoS level can be set which applies to all messages
- Currently the module only supports unsecured MQTT with username/password or secure MQTT/TLS with certificates (root CA, client certificate, and private key). The module does not accept username/password with certificates.

## Examples

### Basic Example

This is a basic example of minimal use of MQTT Middleware in an application.

```
void mqtt_onchip_basic_example (void)
{
 WIFIReturnCode_t wifi_err;
 fsp_err_t       mqtt_err;
 /* Setup Access Point connection parameters */
 WIFINetworkParams_t net_params =
    {
       .ucChannel                  = 0,
       .xPassword.xWPA.cPassphrase = WIFI_PWD,
       .ucSSID                     = WIFI_SSID,
       .xPassword.xWPA.ucLength = sizeof(WIFI_PWD),
       .ucSSIDLength            = sizeof(WIFI_SSID),
       .xSecurity               = eWiFiSecurityWPA2,
    };
 /* Open connection to the Wifi Module */
    wifi_err = WIFI_On();
    assert(eWiFiSuccess == wifi_err);
 /* Connect to the Access Point */
    wifi_err = WIFI_ConnectAP(&net_params);
    assert(eWiFiSuccess == wifi_err);
 /* Initialize the MQTT Client connection */
    mqtt_err = RM_MQTT_DA16XXX_Open(&g_rm_mqtt_onchip_da16xxx_instance,
&g_mqtt_onchip_da16xxx_cfg);
    assert(FSP_SUCCESS == mqtt_err);
 /* Subscribe to MQTT topics to be received */
    mqtt_err = RM_MQTT_DA16XXX_Subscribe(&g_rm_mqtt_onchip_da16xxx_instance,
subTopics, 1);
    assert(FSP_SUCCESS == mqtt_err);
 /* Connect to the MQTT Broker */
    mqtt_err = RM_MQTT_DA16XXX_Connect(&g_rm_mqtt_onchip_da16xxx_instance,
CONNECT_TIMEOUT);
    assert(FSP_SUCCESS == mqtt_err);
 /* Publish data to the MQTT Broker */
    mqtt_err = RM_MQTT_DA16XXX_Publish(&g_rm_mqtt_onchip_da16xxx_instance,
```

```
&pubTopics[0]);
    assert(FSP_SUCCESS == mqtt_err);
 /* Loop to receive data from the MQTT Broker */
 do
    {
 RM_MQTT_DA16XXX_Receive(&g_rm_mqtt_onchip_da16xxx_instance);
 R_BSP_SoftwareDelay(CONNECT_TIMEOUT, BSP_DELAY_UNITS_MILLISECONDS);
    } while (cb_flag == 0);
 /* Disconnect from the MQTT Broker */
    mqtt_err = RM_MQTT_DA16XXX_Disconnect(&g_rm_mqtt_onchip_da16xxx_instance);
    assert(FSP_SUCCESS == mqtt_err);
 /* Close the MQTT Client module */
    mqtt_err = RM_MQTT_DA16XXX_Close(&g_rm_mqtt_onchip_da16xxx_instance);
    assert(FSP_SUCCESS == mqtt_err);
}
```

## Data Structures

| | |
|---|---|
| struct | mqtt_onchip_da16xxx_sub_info_t |
| struct | mqtt_onchip_da16xxx_pub_info_t |
| struct | mqtt_onchip_da16xxx_callback_args_t |
| struct | mqtt_onchip_da16xxx_cfg_t |
| struct | mqtt_onchip_da16xxx_instance_ctrl_t |

## Macros

| | |
|---|---|
| #define | MQTT_ONCHIP_DA16XXX_MAX_ALPN<br>Maximum number of ALPNs supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_MAX_SNI_LEN<br>Maximum length of SNI supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_TLS_CIPHER_SUITE_MAX<br>Maximum number of TLS cipher suites supported by DA16XXX. |

| | |
|---|---|
| #define | MQTT_ONCHIP_DA16XXX_TLS_CIPHER_MAX_CNT<br>Maximum number of TLS cipher suites supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_MAX_TOPIC_LEN<br>Maximum total length for topics supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_MAX_PUBMSG_LEN<br>Maximum total length for message supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_MAX_PUBTOPICMSG_LEN<br>Maximum total length for message + topic supported by DA16XXX. |
| #define | MQTT_ONCHIP_DA16XXX_SUBTOPIC_MAX_CNT<br>Maximum number of subscription topics allowed. |

### Enumerations

| | |
|---|---|
| enum | mqtt_onchip_da16xxx_qos_t |
| enum | mqtt_onchip_da16xxx_tls_cipher_suites_t |

## Data Structure Documentation

### ◆ mqtt_onchip_da16xxx_sub_info_t

| struct mqtt_onchip_da16xxx_sub_info_t | | |
|---|---|---|
| MQTT SUBSCRIBE packet parameters | | |
| Data Fields | | |
| mqtt_onchip_da16xxx_qos_t | qos | Quality of Service for subscription. |
| const char * | p_topic_filter | Topic filter to subscribe to. |
| uint16_t | topic_filter_length | Length of subscription topic filter. |

### ◆ mqtt_onchip_da16xxx_pub_info_t

| struct mqtt_onchip_da16xxx_pub_info_t | | |
|---|---|---|
| MQTT PUBLISH packet parameters | | |
| Data Fields | | |
| mqtt_onchip_da16xxx_qos_t | qos | Quality of Service for |

| | | |
|---|---|---|
| | | subscription. |
| const char * | p_topic_name | Topic name on which the message is published. |
| uint16_t | topic_name_Length | Length of topic name. |
| const char * | p_payload | Message payload. |
| uint32_t | payload_length | Message payload length. |

### ◆ mqtt_onchip_da16xxx_callback_args_t

| struct mqtt_onchip_da16xxx_callback_args_t | | |
|---|---|---|
| MQTT Packet info structure to be passed to user callback | | |
| Data Fields | | |
| uint8_t * | p_data | Payload received from subscribed MQTT topic. |
| const char * | p_topic | Topic to which the message payload belongs to. |
| uint32_t | data_length | Length of the MQTT payload. |
| void * | p_context | Placeholder for user data. |

### ◆ mqtt_onchip_da16xxx_cfg_t

| struct mqtt_onchip_da16xxx_cfg_t | |
|---|---|
| MQTT Configuration | |
| **Data Fields** | |
| const uint8_t | use_mqtt_v311 |
| | Flag to use MQTT v3.1.1. |
| | |
| const uint16_t | rx_timeout |
| | MQTT Rx timeout in milliseconds. |
| | |
| const uint16_t | tx_timeout |
| | MQTT Tx timeout in milliseconds. |
| | |
| void(* | p_callback )(mqtt_onchip_da16xxx_callback_args_t *p_args) |
| | Location of user callback. |
| | |

| | |
|---|---|
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in mqtt_onchip_da16xxx_callback_args_t. |
| | |
| uint8_t | clean_session |
| | Whether to establish a new, clean session or resume a previous session. |
| | |
| uint8_t | alpn_count |
| | ALPN Protocols count. Max value is 3. |
| | |
| const char * | p_alpns [MQTT_ONCHIP_DA16XXX_MAX_ALPN] |
| | ALPN Protocols. |
| | |
| uint8_t | tls_cipher_count |
| | TLS Cipher suites count. Max value is 17. |
| | |
| uint16_t | keep_alive_seconds |
| | MQTT keep alive period. |
| | |
| const char * | p_client_identifier |
| | MQTT Client identifier. Must be unique per client. |
| | |
| uint16_t | client_identifier_length |
| | Length of the client identifier. |
| | |
| const char * | p_host_name |
| | MQTT endpoint host name. |
| | |

| | |
|---|---|
| uint16_t | mqtt_port |
| | MQTT Port number. |
| | |
| const char * | p_mqtt_user_name |
| | MQTT user name. Set to NULL if not used. |
| | |
| uint16_t | user_name_length |
| | Length of MQTT user name. Set to 0 if not used. |
| | |
| const char * | p_mqtt_password |
| | MQTT password. Set to NULL if not used. |
| | |
| uint16_t | password_length |
| | Length of MQTT password. Set to 0 if not used. |
| | |
| const char * | p_root_ca |
| | String representing a trusted server root certificate. |
| | |
| uint32_t | root_ca_size |
| | Size associated with root CA Certificate. |
| | |
| const char * | p_client_cert |
| | String representing a Client certificate. |
| | |
| uint32_t | client_cert_size |
| | Size associated with Client certificate. |
| | |
| const char * | p_client_private_key |

| | |
|---|---|
| | String representing Client Private Key. |
| | |
| uint32_t | private_key_size |
| | Size associated with Client Private Key. |
| | |
| const char * | p_will_topic |
| | String representing Will Topic. |
| | |
| const char * | p_will_msg |
| | String representing Will Message. |
| | |
| const char * | p_sni_name |
| | Server Name Indication. |
| | |
| mqtt_onchip_da16xxx_qos_t | will_qos_level |
| | Will Topic QoS level. |
| | |
| mqtt_onchip_da16xxx_tls_cipher_suites_t | p_tls_cipher_suites [MQTT_ONCHIP_DA16XXX_TLS_CIPHER_MAX_CNT] |
| | TLS Cipher suites supported. |
| | |

### ◆ mqtt_onchip_da16xxx_instance_ctrl_t

| struct mqtt_onchip_da16xxx_instance_ctrl_t | | |
|---|---|---|
| MQTT_ONCHIP_DA16XXX private control block. DO NOT MODIFY. | | |
| Data Fields | | |
| uint8_t | cmd_tx_buff[MQTT_ONCHIP_DA16XXX_CFG_CMD_TX_BUF_SIZE] | Command send buffer. |
| uint8_t | cmd_rx_buff[MQTT_ONCHIP_DA16XXX_CFG_CMD_RX_BUF_SIZE] | Command receive buffer. |

| | | |
|---|---|---|
| uint32_t | open | Flag to indicate if MQTT has been opened. |
| bool | is_mqtt_connected | Flag to track MQTT connection status. |
| mqtt_onchip_da16xxx_cfg_t const * | p_cfg | Pointer to p_cfg for MQTT. |

## Enumeration Type Documentation

### ◆ mqtt_onchip_da16xxx_qos_t

| enum mqtt_onchip_da16xxx_qos_t | |
|---|---|
| MQTT Quality-of-service (QoS) levels | |
| Enumerator | |
| MQTT_ONCHIP_DA16XXX_QOS_0 | Delivery at most once. |
| MQTT_ONCHIP_DA16XXX_QOS_1 | Delivery at least once. |
| MQTT_ONCHIP_DA16XXX_QOS_2 | Delivery exactly once. |

### ◆ mqtt_onchip_da16xxx_tls_cipher_suites_t

| enum mqtt_onchip_da16xxx_tls_cipher_suites_t | |
|---|---|
| MQTT TLS Cipher Suites | |
| Enumerator | |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA protocol. |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA protocol. |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 protocol. |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 protocol. |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 protocol. |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 protocol. |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 protocol. |

## Function Documentation

### ◆ RM_MQTT_DA16XXX_Open()

fsp_err_t RM_MQTT_DA16XXX_Open ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, mqtt_onchip_da16xxx_cfg_t const *const *p_cfg* )

Initialize the DA16XXX on-chip MQTT Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |
| [in] | p_cfg | Pointer to MQTT Client configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_cfg instance is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Data size is too large or NULL. |
| FSP_ERR_ALREADY_OPEN | The instance has already been opened. |

### ◆ RM_MQTT_DA16XXX_Disconnect()

fsp_err_t RM_MQTT_DA16XXX_Disconnect ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*)

Disconnect from DA16XXX MQTT Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_ctrl instance is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened or the client is not connected. |

### ◆ RM_MQTT_DA16XXX_Connect()

fsp_err_t RM_MQTT_DA16XXX_Connect ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, uint32_t *timeout_ms* )

Configure and connect the DA16XXX MQTT Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |
| [in] | timeout_ms | Timeout in milliseconds. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_IN_USE | The MQTT client is already connected. |
| FSP_ERR_INVALID_DATA | Response does not contain Connect status. |

### ◆ RM_MQTT_DA16XXX_Publish()

fsp_err_t RM_MQTT_DA16XXX_Publish ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, mqtt_onchip_da16xxx_pub_info_t *const *p_pub_info* )

Publish a message for a given MQTT topic.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |
| [in] | p_pub_info | MQTT Publish packet parameters. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_ctrl, p_pub_info is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened or the client is not connected. |
| FSP_ERR_INVALID_DATA | Data size is too large. |

### ◆ RM_MQTT_DA16XXX_Subscribe()

fsp_err_t RM_MQTT_DA16XXX_Subscribe ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, mqtt_onchip_da16xxx_sub_info_t *const *p_sub_info*, size_t *subscription_count* )

Subscribe to DA16XXX MQTT topics.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |
| [in] | p_sub_info | List of MQTT subscription info. |
| [in] | subscription_count | Number of topics to subscribe to. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_ctrl, p_sub_info is NULL or subscription_count is 0. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_DATA | Data size is too large. |

### ◆ RM_MQTT_DA16XXX_UnSubscribe()

fsp_err_t RM_MQTT_DA16XXX_UnSubscribe ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*, mqtt_onchip_da16xxx_sub_info_t *const *p_sub_info* )

Unsubscribe from DA16XXX MQTT topics.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |
| [in] | p_sub_info | List of MQTT subscription info. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_ctrl, p_sub_info is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened or the client is not connected. |
| FSP_ERR_INVALID_DATA | Data size is too large. |

### ◆ RM_MQTT_DA16XXX_Receive()

fsp_err_t RM_MQTT_DA16XXX_Receive ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*)

Receive data subscribed to on DA16XXX MQTT Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The p_ctrl, p_textstring, p_ip_addr is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened or the client is not connected. |
| FSP_ERR_INVALID_DATA | Receive function did not receive valid publish data. |

### ◆ RM_MQTT_DA16XXX_Close()

fsp_err_t RM_MQTT_DA16XXX_Close ( mqtt_onchip_da16xxx_instance_ctrl_t * *p_ctrl*)

Close the DA16XXX MQTT Client service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MQTT Client instance control structure. |

**Return values**

| | |
|---|---|
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The p_ctrl, p_textstring, p_ip_addr is NULL. |

## 5.2.12.29 PTP (r_ptp)

Modules » Networking

**Functions**

| | |
|---|---|
| fsp_err_t | R_PTP_Open (ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg) |
| fsp_err_t | R_PTP_MacAddrSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr) |
| fsp_err_t | R_PTP_IpAddrSet (ptp_ctrl_t *const p_ctrl, uint32_t ip_addr) |
| fsp_err_t | R_PTP_LocalClockIdSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id) |
| fsp_err_t | R_PTP_MasterClockIdSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id) |
| fsp_err_t | R_PTP_MessageFlagsSet (ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags) |
| fsp_err_t | R_PTP_CurrentUtcOffsetSet (ptp_ctrl_t *const p_ctrl, uint16_t offset) |
| fsp_err_t | R_PTP_PortStateSet (ptp_ctrl_t *const p_ctrl, uint32_t state) |
| fsp_err_t | R_PTP_MessageSend (ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size) |

| | |
|---|---|
| fsp_err_t | R_PTP_LocalClockValueSet (ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time) |
| fsp_err_t | R_PTP_LocalClockValueGet (ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time) |
| fsp_err_t | R_PTP_PulseTimerCommonConfig (ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *const p_timer_cfg) |
| fsp_err_t | R_PTP_PulseTimerEnable (ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg) |
| fsp_err_t | R_PTP_PulseTimerDisable (ptp_ctrl_t *const p_ctrl, uint32_t channel) |
| fsp_err_t | R_PTP_Close (ptp_ctrl_t *const p_ctrl) |
| fsp_err_t | R_PTP_BestMasterClock (ptp_message_t const *const p_announce1, ptp_message_t const *const p_announce2, int8_t *const p_comparison) |

### Detailed Description

Driver for the PTP peripheral on RA MCUs. This module implements the PTP Interface.

# Overview

PTP allows for multiple devices on a network to synchronize their clocks with very high precision. The PTP peripheral generates and processes PTP messages automatically. In slave mode, it also corrects the local time in order to adjust for any offset from the master clock time.

### Features

- Ordinary clock
  - Master mode
  - Slave mode
- Peer-to-peer
- End-to-end
- Frame fromats
  - Ethernet II frames
  - IEEE802.3 + LLC + SNAP frames
  - IPv4 + UDP
- Clock correction modes
  - Mode 1: Add the offsetFromMaster value to the local time whenever it is updated.
  - Mode 2: Calculate a clock gradient and continuously adjust the local time in order to minimize the offsetFromMaster value.

# Configuration

### Build Time Configurations for r_ptp

The following build time configurations are defined in fsp_cfg/r_ptp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Networking > PTP (r_ptp)**

This module can be added to the Stacks tab via New Stack > Networking > PTP (r_ptp).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Clock Properties | | | |
| Priority 1 | Value must in the range [0,255]. | 128 | Priority1 field advertised in generated announce packets. |
| Class | Value must in the range [0,255]. | 248 | Class field advertised in generated announce packets. |
| Accuracy | Value must in the range [0,255]. | 0xFE | Accuracy field advertised in generated announce packets. |
| Variance | Value must in the range [0,65535]. | 0xFFFF | Variance field advertised in generated announce packets. |
| Priority 2 | Value must in the range [0,255]. | 128 | Priority2 field advertised in generated announce packets. |
| Time Source | Value must in the range [0,255]. | 160 | Time Source field advertised in generated announce packets. |
| Ethernet | | | |
| Multicast Filter MAC address | Must be a valid MAC address | 01:1B:19:00:00:00 | In Multicast Filtered mode, only multicast addresses that match this address are received by the ETHERC EDMAC. |
| Primary Destination MAC address | Must be a valid MAC address | 01:1B:19:00:00:00 | The destination MAC address for primary PTP messages. |
| PDelay Destination MAC address | Must be a valid MAC address | 01:80:C2:00:00:0E | The destination MAC address for PDelay messages. |

| | | | |
|---|---|---|---|
| IP | | | |
| Primary Destination IP address | Must be a valid IP address | 224.0.1.129 | The destination IPv4 address for primary messages. |
| PDelay Destination IP address | Must be a valid IP address | 224.0.0.107 | The destination IPv4 address for PDelay messages. |
| Event Message TOS | Value must in the range [0,255]. | 0 | The IP packet TOS for event messages. |
| General Message TOS | Value must in the range [0,255]. | 0 | The IP packet TOS for general messages. |
| Primary Message TTL | Value must in the range [0,255]. | 1 | The IP packet TTL for primary messages. |
| PDelay Message TTL | Value must in the range [0,255]. | 1 | The IP packet TTL for p_delay messages. |
| Event Port | Value must in the range [0,65535]. | 319 | The UDP port for event messages. |
| General Port | Value must in the range [0,65535]. | 320 | The UDP port for general messages. |
| Synchronization Detection | | | |
| Threshold (Nanoseconds) | Value must be greater than 0. | 1000000 | The minimum offsetFromMaster value required in order to synchronize with the master clock. |
| Count | Value must in the range [0,15]. | 5 | The number of times the calculated offsetFromMaster value must be less than the threshold in order to synchronize with the master clock. |
| Synchronization Lost Detection | | | |
| Threshold (Nanoseconds) | Value must be greater than 0. | 10000000 | The minimum offsetFromMaster value required in order to lose synchronization with the master clock. |
| Count | Value must in the range [0,15]. | 5 | The number of times the calculated offsetFromMaster value must be greater than the threshold in order to lose synchronization with the master. |

| | | | |
|---|---|---|---|
| Interrupts | | | |
| Callback | Name must be a valid C symbol | ptp0_user_callback | Called when a STCA/SYNFP event occurs, a PTP message is received, or if a Pulse Timer event occurs. |
| MINT Interrupt priority | MCU Specific Options | | Select the EPTPC MINT interrupt priority. |
| Pulse Timer Interrupt priority | MCU Specific Options | | Select the EPTPC IPLS priority. |
| Name | Name must be a valid C symbol | g_ptp0 | Module name. |
| Ethernet PHY Interface Type | • MII<br>• RMII | RMII | The interface type used to communicate with the Ethernet PHY. |
| Frame Filter | • Extended Promiscuous Mode<br>• Unicast and Multicast<br>• Unicast and Multicast Filtered<br>• Unicast | Unicast | Selects how packets are filtered based on their destination MAC address. Packets that pass the filter are transferred to the ETHERC EDMAC. |
| Frame Format | • Ethernet II<br>• Ethernet II \| IPv4 \| UDP<br>• IEEE802.3 \| LLC \| SNAP<br>• IEEE802.3 \| LLC \| SNAP \| IPv4 \| UDP | Ethernet II | The format of the frames that encapsulate the PTP messages. |
| Clock Domain | Value must in the range [0,255]. | 0 | The PTP clock will only respond to clocks in its domain. |
| Clock Domain Filter | • Enable<br>• Disable | Enable | Filter out PTP messages from other clock domains. |
| Buffer Size | Value must in the range [64,1536]. | 1536 | The maximum Ethernet packet size that can be transmitted or received by the application from the EDMAC. |
| Number of transmit buffers | Value must in the range [1,16]. | 4 | The number of transmit buffers in the packet queue. |
| Number of receive | Value must in the | 4 | The number of receive |

| | | | |
|---|---|---|---|
| buffers | range [1,16]. | | buffers in the packet queue. |
| Announce message interval. | MCU Specific Options | | The period of time between generated announce messages. |
| Sync message interval. | MCU Specific Options | | The period of time between generated sync messages. |
| Delay_req message interval. | MCU Specific Options | | The period of time between generated delay_req messages. |
| Message timeout | Value must be greater than 0. | 4000 | The time in milliseconds needed to generate timeout events after not receiving a sync or delay_resp message. |
| Clock Source | • PCLKA / 1<br>• PCLKA / 2<br>• PCLKA / 3<br>• PCLKA / 4<br>• PCLKA / 5<br>• PCLKA / 6<br>• REF50CK0 | PCLKA / 6 | The STCA clock source must be 20Mhz, 25Mhz, 50Mhz, or 100Mhz. When REF50CK0 is selected, the STCA frequency is 25Mhz. |
| Clock Correction Mode | • Clock Correction Mode 1<br>• Clock Correction Mode 2 | Clock Correction Mode 1 | Clock correction mode 1 corrects the local clock using the current offsetFromMaster value. Clock correction mode 2 calculates a clock gradient in order to continuously correct the local clock. |
| Gradient Worst10 Interval | Value must in the range [0,255]. | 32 | The number of sync messages to use when calculating the worst10 gradient values (Only applies to clock correction mode 2). |

### Clock Configuration

The STCA input clock can be the following clock sources:

- PCLKA / 1
- PCLKA / 2
- PCLKA / 3
- PCLKA / 4
- PCLKA / 5
- PCLKA / 6
- REF50CK0

The STCA input clock is restricted to the following frequencies:

- 20 Mhz
- 25 Mhz
- 50 Mhz
- 100 Mhz

When REF50CK0 is selected, the input clock frequency is 25 Mhz.

### Pin Configuration

The PTP module requires the Ethernet (r_ether) instance in order to initialize the Ethernet PHY. This means that the ETHERC pins must be configured.

# Usage Notes

### PTP Port State

The current PTP port state determines which messages need to be generated and processed by the PTP peripheral. It is the application's responsibility to determine what the current state of the PTP port should be.

The following messages can be generated by the PTP peripheral:

- Announce
- Sync
- Delay_req
- Delay_resp
- PDelay_req
- PDelay_resp

The following messages can be processed by the PTP peripheral:

- Sync
- Follow_up
- Delay_req
- Delay_resp
- PDelay_req
- PDelay_resp
- PDelay_resp_follow_up

The application must receive the following messages in order to determine the current state of its PTP port:

- Announce
- Management
- Signaling

The following messages can only be sent manually:

- Management
- Signaling

The PTP API defines the following states:

| State | Generated Messages | Processed Messages | Received Messages |
|---|---|---|---|
| Disabled | N/A | N/A | N/A |
| Passive | N/A | N/A | Announce, Signaling, Management |
| E2E/P2P Slave | Delay_req/(PDelay_req, PDelay_resp) | Sync, Follow_up, Delay_resp/(PDelay_req, PDelay_resp) | Announce, Signaling, Management |
| E2E/P2P Master | Announce, Sync, Delay_resp/(PDelay_req, PDelay_resp) | delay_req/(PDelay_req, PDelay_resp) | Announce, Signaling, Management |

### Pulse Timers

Pulse Timers are configurable timers used to generate interrupts and ELC events. Each pulse timer has a configurable start time, pulse, and period. At the start of each timer period, a rising edge occurs. After the pulse time has elapsed, a falling edge occurs. ELC events and IRQs can be generated on rising and/or falling edges for each Pulse Timer. There are two types of interrupts generated by each Pulse Timer; MINT and IPLS.

### ELC Events

Pulse timers may be configured to generate the following ELC events:

- EPTPC_TIMERn_RISE - Generated on the rising edge of pulse timer channel n[1].
- EPTPC_TIMERn_FALL - Generated on the falling edge of pulse timer channel n[1].

*Note*

*1. n = [0,5] corresponds to the channel of the pulse timer.*

### MINT Interrupts

MINT IRQs are only generated on the rising edge of a Pulse Timer channel. The callback will provide the channel number of the pulse timer that caused the interrupt.

### IPLS Interrupts

Each Pulse Timer channel can be configured as a source for generating IPLS IRQs. All of the pulse timers that are selected as IPLS sources are OR'd together and rising and falling edge IRQs can be generated from the resulting signal. Below is an example of a resulting signal from two IPLS sources. Unlike MINT interrupts, IPLS interrupts do not provide any information about which Pulse Timer caused the IRQ because the IRQs from all the Pulse Timers are OR'd together.

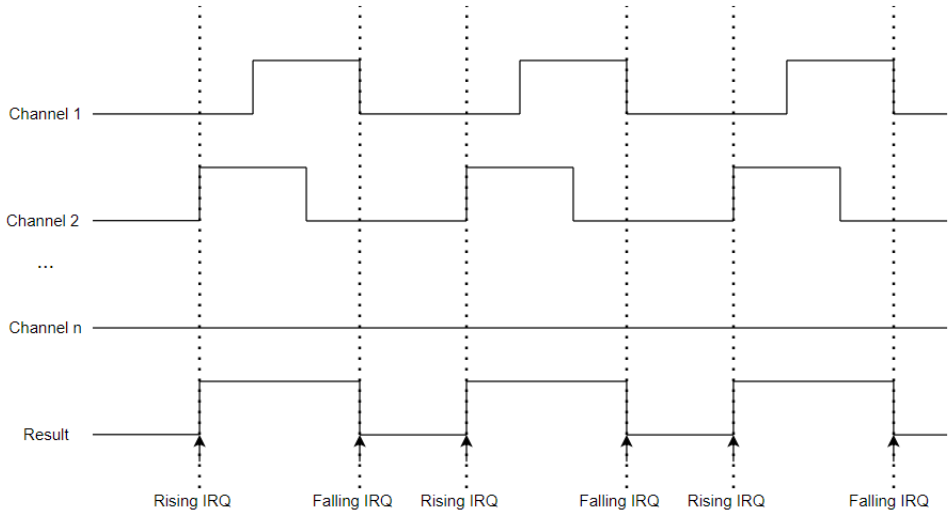



Figure 326: IPLS IRQ Generation

### Ethernet Frame Filter

The PTP driver can filter Ethernet frames that are received by Ethernet (r_ether). There are four different filtering modes:

- Extended Promiscuous - All Ethernet frames are received by Ethernet (r_ether).
- Unicast and Multicast - All Unicast frames destined for the PTP and Multicast frames are received by Ethernet (r_ether).
- Unicast and Multicast Filtered - All Unicast frames destined for the PTP are received by Ethernet (r_ether). All multicast frames that match ptp_synfp_cfg_t::p_multicast_addr_filter are received by Ethernet (r_ether).
- Unicast - Only Unicast frames destined for the PTP are received by Ethernet (r_ether).

### Limitations

Developers should be aware of the following limitations when using the PTP:

- PTP will not automatically initialize Ethernet (r_ether). This provides flexibility by allowing PTP to be used alongside 3rd party IP stacks (Eg. FreeRTOS+TCP Wrapper to r_ether (rm_freertos_plus_tcp)), however this means the application must execute the Ethernet (r_ether) link process in order to use PTP.
- The driver will not detect announce message timeouts. This functionality must be handled by the application.
- When IP + UDP frame format is selected, the driver will not automatically join the multicast group. This must be done by the application.
- In order to call PTP API functions from ISRs, the MINT and IPLS interrupt priorities must be configured to be lower than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. This is to guarantee that PTP register accesses are atomic.
- When IEEE802.3 | LLC | SNAP | IP | UDP or IEEE802.3 | LLC | SNAP Frame Format is selected, the PTP peripheral sets the OUI field in the SNAP frame to the first 3 bytes of the **MAC address** configured in the Ethernet (r_ether) module's configuration property. In order for the PID field of SNAP to be interpreted as EtherType, the OUI field should be set to '0'.

# Examples

### Slave Mode

This is a basic example of minimal use of PTP in slave mode.

```
volatile bool g_first_announce_message_received = false;
volatile bool g_sync_acquired = false;
void slave_mode_basic_example (void)
{
 /* The PTP Instance must be opened before R_ETHER is opened. */
 fsp_err_t err = R_PTP_Open(&g_ptp_ctrl, &g_ptp_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP MAC address. */
    err = R_PTP_MacAddrSet(&g_ptp_ctrl, g_ptp_mac_address);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP Local Clock ID (Usually generated from MAC address). */
    err = R_PTP_LocalClockIdSet(&g_ptp_ctrl, g_ptp_clock_id);
    assert(FSP_SUCCESS == err);
 /* Open the r_ether_api instance. */
    err = R_ETHER_Open(&g_ether_ctrl, &g_ether_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the link to be established. */
 do
    {
       err = R_ETHER_LinkProcess(&g_ether_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set the PTP instance to passive state and listen for announce message. */
    err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_PASSIVE);
    assert(FSP_SUCCESS == err);
 /* Wait for the first announce message (This will provide the master clock ID). */
    uint32_t timeout = EXAMPLE_TIMEOUT;
 while (!g_first_announce_message_received && --timeout)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
    assert(0U != timeout);
 /* When a master clock is found, change to the slave state to start synchronizing
```

```
 * the local clock to the master clock. */
   err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_E2E_SLAVE);
   assert(FSP_SUCCESS == err);
/* Wait for local clock to be synchronized with the master clock. */
   timeout = EXAMPLE_TIMEOUT;
while (!g_sync_acquired && --timeout)
   {
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
   }
   assert(0U != timeout);
/* The local clock is now synchronized with the grand master clock. */
}
/* Callback called whenever a PTP event occurs. */
void g_ptp_slave_callback_example (ptp_callback_args_t * p_args)
{
switch (p_args->event)
   {
case PTP_EVENT_SYNC_ACQUIRED:
     {
/* The offsetFromMaster value is now within the configured threshold to be
 * synchronized with the master clock.
 */
           g_sync_acquired = true;
break;
     }
case PTP_EVENT_MESSAGE_RECEIVED:
     {
static ptp_message_t g_current_master_announce_message;
switch (p_args->p_message->header.message_type)
     {
case PTP_MESSAGE_TYPE_ANNOUNCE:
     {
                int8_t comparison = 0;
if (!g_first_announce_message_received)
```

```
     {
 /* If this is the first announce packet, immediately switch to this master clock. */
                       comparison = 1;
                       g_first_announce_message_received = true;
     }
 else
     {
 /*
  * Run the "Best Master Clock Algorithm" to determine if the clock defined in this
announce
  * packet is better than the current master clock.
  */
 fsp_err_t err = R_PTP_BestMasterClock(&g_current_master_announce_message,
                                                        p_args->p_message,
                                                        &comparison);
     assert(FSP_SUCCESS == err);
     }
 if (1 == comparison)
     {
 /* Save the message as the new master announce message. */
                       g_current_master_announce_message = *p_args->p_message;
 /* Set the master clock ID and sourcePortID in the PTP instance so that it
  * synchronizes with the new best master clock.
  */
 fsp_err_t err = R_PTP_MasterClockIdSet(&g_ptp_ctrl,
                                                          g_current_master_annou
nce_message.header.clock_id,
                                                          g_current_master_annou
nce_message.header.source_port_id);
     assert(FSP_SUCCESS == err);
     }
 break;
     }
 default:
```

```
     {
 break;
     }
     }
 break;
     }
 default:
     {
 break;
     }
   }
}
```

### Master Mode

This is a basic example of minimal use of PTP in master mode.

```
#define PTP_EXAMPLE_CURRENT_UTC_OFFSET (37)
void master_mode_basic_example (void)
{
 /* The PTP Instance must be opened before R_ETHER is opened. */
 fsp_err_t err = R_PTP_Open(&g_ptp_ctrl, &g_ptp_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP MAC address. */
    err = R_PTP_MacAddrSet(&g_ptp_ctrl, g_ptp_mac_address);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP Local Clock ID (Usually generated from MAC address). */
    err = R_PTP_LocalClockIdSet(&g_ptp_ctrl, g_ptp_clock_id);
    assert(FSP_SUCCESS == err);
 /* Get the current time from an external time source (Eg. RTC). */
 ptp_time_t current_time;
    get_current_time_example(&current_time);
 /* Set the PTP local time to the current time. */
    err = R_PTP_LocalClockValueSet(&g_ptp_ctrl, &current_time);
    assert(FSP_SUCCESS == err);
```

```
 /* Set the currentUtcOffset field in announce messages. */
    err = R_PTP_CurrentUtcOffsetSet(&g_ptp_ctrl, PTP_EXAMPLE_CURRENT_UTC_OFFSET);
    assert(FSP_SUCCESS == err);
 /* Set message flags in announce messages to indicate that the current UTC offset is
valid and that the PTP timescale is used. */
 ptp_message_flags_t flags;
    flags.value = 0;
    flags.value_b.currentUtcOffsetValid = 1;
    flags.value_b.ptpTimescale          = 1;
    err = R_PTP_MessageFlagsSet(&g_ptp_ctrl, PTP_MESSAGE_TYPE_ANNOUNCE, flags);
    assert(FSP_SUCCESS == err);
 /* Open the r_ether_api instance. */
    err = R_ETHER_Open(&g_ether_ctrl, &g_ether_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the link to be established. */
 do
    {
       err = R_ETHER_LinkProcess(&g_ether_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set the PTP instance to passive state and listen for announce message. */
    err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_E2E_MASTER);
    assert(FSP_SUCCESS == err);
 /*
  * The master clock is now operational and will automatically generate announce and
sync messages
  * as well as respond to delay_req messages.
  */
}
```

### Send PTP Messages

This is a basic example of how to send PTP messages.

```
#define PTP_MANAGEMENT_ACTION_GET (0U)
#define PTP_TLV_TYPE_MANAGEMENT (1U)
```

```
#define PTP_TLV_MANAGEMENT_ID_CLOCK_DESCRIPTION (1U)
static uint32_t g_transmit_complete = 0U;
void send_message_example (void)
{
 static ptp_message_t message;
 static uint8_t       p_tlv_data[6];
    memset(&message, 0, sizeof(ptp_message_t));
 /* Fill in the required fields for the message header (Note that appropriate fields
will be endian swapped). */
    message.header.message_type = PTP_MESSAGE_TYPE_MANAGEMENT;
    message.header.version      = 2;
 /* The message length is the total number of bytes in the PTP message (Including the
message header). */
    message.header.message_length = (uint16_t) (sizeof(ptp_message_header_t) +
 sizeof(ptp_message_management_t) +
 sizeof(p_tlv_data));
    memcpy(message.header.clock_id, g_ptp_clock_id, sizeof(g_ptp_clock_id));
    message.header.control_field = PTP_CTRL_FIELD_MANAGEMENT;
 /* Fill in the required fields for the management message. */
    memcpy(message.management.target_clock_id, g_target_clock_id, sizeof
(g_target_clock_id));
    message.management.target_port_id         = 1;
    message.management.starting_boundary_hops = 1;
    message.management.boundary_hops          = 1;
    message.management.action                = PTP_MANAGEMENT_ACTION_GET;
 /*
  * Fill in TLV data (Note that TLV data is big endian).
  *
  * Type (Management)
  */
    p_tlv_data[0] = 0;
    p_tlv_data[1] = PTP_TLV_TYPE_MANAGEMENT;
 /* Length */
    p_tlv_data[2] = 0;
```

```
    p_tlv_data[3] = 2;
 /* Management ID (Clock Description) */
    p_tlv_data[4] = 0;
    p_tlv_data[5] = PTP_TLV_MANAGEMENT_ID_CLOCK_DESCRIPTION;
 /* Send the message. */
 fsp_err_t err = R_PTP_MessageSend(&g_ptp_ctrl, &message, p_tlv_data, sizeof
(p_tlv_data));
    assert(FSP_SUCCESS == err);
    uint32_t timeout = EXAMPLE_TIMEOUT;
 while (0U == g_transmit_complete && --timeout)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MICROSECONDS);
    }
}
/* Callback called whenever a PTP event occurs. */
void g_ptp_send_message_callback_example (ptp_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case PTP_EVENT_MESSAGE_TRANSMIT_COMPLETE:
     {
           g_transmit_complete = 1U;
 break;
     }
 case PTP_EVENT_MESSAGE_RECEIVED:
     {
 switch (p_args->p_message->header.message_type)
     {
 case PTP_MESSAGE_TYPE_MANAGEMENT:
     {
 /* Handle the response message. */
 break;
     }
 default:
```

```
      {
 break;
      }
      }
      }
 default:
      {
 break;
      }
    }
}
```

### Data Structures

| | |
|---|---|
| struct | ptp_instance_ctrl_t |

## Data Structure Documentation

### ◆ ptp_instance_ctrl_t

| struct ptp_instance_ctrl_t | | |
|---|---|---|
| PTP instance control block. | | |
| Data Fields | | |
| uint32_t | open | Marks if the instance has been opened. |
| uint32_t | tx_buffer_write_index | Index into the descriptor list to write the next packet. |
| uint32_t | tx_buffer_complete_index | Index into the descriptor list of the last transmitted packet. |
| uint32_t | rx_buffer_index | Index into the descriptor of the last received packet. |
| uint32_t | tslatr | Keep track of whether tslatr was set. |
| ptp_cfg_t const * | p_cfg | Pointer to the configuration structure. |

## Function Documentation

### ◆ R_PTP_Open()

fsp_err_t R_PTP_Open ( ptp_ctrl_t *const *p_ctrl*, ptp_cfg_t const *const *p_cfg* )

This function initializes PTP. Implements ptp_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the peripheral registers according to the configuration.
- Initialize the control structure for use in other PTP Interface functions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The instance has been successfully configured. |
| FSP_ERR_ALREADY_OPEN | Instance was already initialized. |
| FSP_ERR_NOT_OPEN | The EDMAC instance was not opened correctly. |
| FSP_ERR_ASSERTION | An invalid argument was given in the configuration structure. |

### ◆ R_PTP_MacAddrSet()

fsp_err_t R_PTP_MacAddrSet ( ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac_addr* )

This function sets the MAC address for the PTP instance. Implements ptp_api_t::macAddrSet.

***Note***

*This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The MAC address has been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_INVALID_MODE | The instance is not in the correct state. |

### ◆ R_PTP_IpAddrSet()

fsp_err_t R_PTP_IpAddrSet ( ptp_ctrl_t *const *p_ctrl*, uint32_t *ip_addr* )

This function sets the IP address for the PTP instance. Implements ptp_api_t::ipAddrSet.

***Note***

*This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The IP address has been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL. |
| FSP_ERR_INVALID_MODE | The configured ptp_synfp_cfg_t::frame_format is not configured to use IP packets, or the instance is not in the correct state. |

### ◆ R_PTP_LocalClockIdSet()

fsp_err_t R_PTP_LocalClockIdSet ( ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_clock_id* )

This function sets the local clock ID for the PTP instance. Implements ptp_api_t::localClockIdSet.

***Note***

*This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.*
*Typically the clock ID is derived from the MAC address (E.g. {b1,b2,b3,0xFF,0xFE,b4,b5,b6}).*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The local clock ID has been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL |
| FSP_ERR_INVALID_MODE | The instance is not in the correct state. |

### ◆ R_PTP_MasterClockIdSet()

fsp_err_t R_PTP_MasterClockIdSet ( ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_clock_id*, uint16_t *port_id* )

This function sets the master clock ID and port ID that the local clock will synchronize with. Implements ptp_api_t::masterClockIdSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The master clock ID and port ID have been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL |

### ◆ R_PTP_MessageFlagsSet()

fsp_err_t R_PTP_MessageFlagsSet ( ptp_ctrl_t *const *p_ctrl*, ptp_message_type_t *message_type*, ptp_message_flags_t *flags* )

This function sets the flags field for the given message type. Implements ptp_api_t::messageFlagsSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The master clock ID and port ID have been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_CurrentUtcOffsetSet()

fsp_err_t R_PTP_CurrentUtcOffsetSet ( ptp_ctrl_t *const *p_ctrl*, uint16_t *offset* )

This function sets the currentUtcOffset value in announce messages. ptp_api_t::currentUtcOffsetSet .

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The currentUtcOffset has been updated. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_PortStateSet()

fsp_err_t R_PTP_PortStateSet ( ptp_ctrl_t *const *p_ctrl*, uint32_t *state* )

This function changes the current state of the PTP instance. Implements ptp_api_t::portStateSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The instance will transition to the new state. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL |

### ◆ R_PTP_MessageSend()

fsp_err_t R_PTP_MessageSend ( ptp_ctrl_t *const *p_ctrl*, ptp_message_t const *const *p_message*, uint8_t const *const *p_tlv_data*, uint16_t *tlv_data_size* )

This function sends a PTP message. ptp_api_t::messageSend.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The packet has been written to the transmit descriptor. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |
| FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL | There is no space for the packet in the transmit queue. |

### ◆ R_PTP_LocalClockValueSet()

fsp_err_t R_PTP_LocalClockValueSet ( ptp_ctrl_t *const *p_ctrl*, ptp_time_t const *const *p_time* )

This function sets the local clock value. Implements ptp_api_t::localClockValueSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The local clock value has been set. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_LocalClockValueGet()

fsp_err_t R_PTP_LocalClockValueGet ( ptp_ctrl_t *const *p_ctrl*, ptp_time_t *const *p_time* )

This function gets the local clock value. Implements ptp_api_t::localClockValueGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The local clock value has been written in p_time. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL |

### ◆ R_PTP_PulseTimerCommonConfig()

fsp_err_t R_PTP_PulseTimerCommonConfig ( ptp_ctrl_t *const *p_ctrl*, ptp_pulse_timer_common_cfg_t *const *p_timer_cfg* )

This function configures IPLS IRQ settings that are common to all pulse timer channels. Implements ptp_api_t::pulseTimerCommonConfig.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The pulse timer has been enabled. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_PulseTimerEnable()

fsp_err_t R_PTP_PulseTimerEnable ( ptp_ctrl_t *const *p_ctrl*, uint32_t *channel*, ptp_pulse_timer_cfg_t *const *p_timer_cfg* )

This function enables a pulse timer channel. Implements ptp_api_t::pulseTimerEnable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The pulse timer has been enabled. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |
| FSP_ERR_INVALID_ARGUMENT | The start time must be set to a value that is later than current time. |

### ◆ R_PTP_PulseTimerDisable()

fsp_err_t R_PTP_PulseTimerDisable ( ptp_ctrl_t *const *p_ctrl*, uint32_t *channel* )

This function disables a pulse timer channel. Implements ptp_api_t::pulseTimerDisable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The pulse timer has been disabled. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_Close()

fsp_err_t R_PTP_Close ( ptp_ctrl_t *const *p_ctrl*)

Disable the PTP instance. Implements ptp_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The pulse timer has been disabled. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | An argument was NULL or invalid. |

### ◆ R_PTP_BestMasterClock()

fsp_err_t R_PTP_BestMasterClock ( ptp_message_t const *const *p_announce1*, ptp_message_t const *const *p_announce2*, int8_t *const *p_comparison* )

This function compares two clocks to determine which one is the better master clock.

p_comparison:

- Set to -1 if p_announce1 defines the best master clock.
- Set to 1 if p_announce2 defines the best master clock.
- Set to 0 if p_announce1 and p_announce2 define the same clock.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The valid result has been written to p_use_announce_clock. |
| FSP_ERR_ASSERTION | An argument was NULL. |

## 5.2.12.30 SPP BLE Abstraction (rm_ble_abs_spp)

Modules » Networking

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

# Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE. QE for BLE provides standard and custom services defined by the user.

The module supports the Renesas Electronics RYZ012 Bluetooth® Low Energy 5 Module. The rm_ble_abs_spp driver interfaces with the RYZ012 module over UART or SPI.

### Features

The Bluetooth Low Energy (BLE) Abstraction module with SPP supports the following features:

- Common functionality
  - Open/Close the BLE protocol stack.
  - Setting Transmit Power Level.
- The following GAP Role support
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
- GAP functionality
  - Initialize the Host stack.
  - Setting address.
  - Start/Stop Advertising.
  - Connect/Disconnect a link.
- GATT Common functionality
  - Get MTU Size.
- GATT Server functionality
  - Initialization of GATT Server.
  - Loading of Profile definition.
  - Notification of characteristics modification.
  - Read/Write of GATT Profile from host.
- Selectable Communication Interfaces
  - SPI Interface
  - UART Interface

### Target Devices

The BLE Abstraction module supports the following devices.

- RA6 line of devices using the RYZ012 module
- RA4 Line of devices using the RYZ012 module

# Configuration

### Build Time Configurations for rm_ble_abs_spp

The following build time configurations are defined in fsp_cfg/rm_ble_abs_spp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enable<br>• Disable | Default (BSP) | Specify whether to include code for API parameter checking. |
| Reset Port | Refer to the RA Configuration tool for available options. | 03 | Specify the module reset pin port for the MCU. |
| Reset Pin | Refer to the RA Configuration tool for available options. | 10 | Specify the module reset pin for the MCU. |
| UART/SPI Select Port (PB5) | Refer to the RA Configuration tool for available options. | 03 | Specify the module PB5 (UART/SPI select) port for the MCU. |
| UART/SPI Select Pin (PB5) | Refer to the RA Configuration tool for available options. | 11 | Specify the module PB5 (UART/SPI select) pin for the MCU. |
| SPI Software SSL Port | Refer to the RA Configuration tool for available options. | Disabled | Specify the port to use for controlling SSL using the ioport module. |
| SPI Software SSL Pin | Refer to the RA Configuration tool for available options. | Disabled | Specify the pin to use for controlling SSL using the ioport module. |
| Transmit Power Level (in dBm) | Refer to the RA Configuration tool for available options. | 4.57 | Specify the module transmit power in dBm. |

**Configurations for Networking > SPP BLE Abstraction (rm_ble_abs_spp)**

This module can be added to the Stacks tab via New Stack > Networking > SPP BLE Abstraction (rm_ble_abs_spp).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_ble_abs0 | Module name. |
| GAP callback | Name must be a valid C symbol | gap_cb | If QE is used, set to "gap_cb". |
| Vendor specific callback | Name must be a valid C symbol | vs_cb | If QE is used, set to "vs_cb". |
| GATT server callback parameter | Name must be a valid C symbol | gs_abs_gatts_cb_param | If QE is used, set to "gs _abs_gattc_cb_param". |
| GATT server callback number | Must be a valid number | 2 | The number of GATT Server callback functions. |

| | | | |
|---|---|---|---|
| GATT client callback parameter | Name must be a valid C symbol | gs_abs_gattc_cb_param | Set GATT client callback parameter. If QE is used, set to "gs_abs_gattc_cb_param". |
| GATT client callback number | Must be a valid number | 2 | The number of GATT Server callback functions. |

### Pin Configuration

Pins used by the BLE driver to control the RYZ012 module:

- Reset Pin : Active low reset line for RYZ012 module
- UART/SPI Select Pin : Allows selection between UART or SPI for module communication. UART is selected with output low and SPI with output high.
- UART Interface : RX/TX Pins
- SPI Interface : MOSI/MISO/SCK/SSL Pins
  - Data Ready Pin : When the SPI Interface is selected as the communication interface, the Data Ready pin is asserted in order to notify the MCU that there is data ready to be read.
  - Software SSL Pin : Can be optionally configured in order to drive the SSL Pin using software instead of hardware[1].

*Note*

*1. The software SSL Pin is required when using the SPI (r_sci_spi) driver.*
*2. When using software SSL, the pin must be configured as "Output mode (Initial High)" in the pin configuration editor in addition to being configured in the "stacks" tab.*

# Usage Notes

### Limitations

Developers should be aware of the following limitations when using the BLE_ABS:

Currently supported rm_ble_abs_spp interface functions:

- RM_BLE_ABS_Open
- RM_BLE_ABS_Close
- RM_BLE_ABS_StartLegacyAdvertising

Currently supported r_ble_spp_api interface functions:

- R_BLE_Open
- R_BLE_Close
- R_BLE_SetEvent
- R_BLE_GAP_Init
- R_BLE_GAP_StopAdv
- R_BLE_GAP_SetAdvSresData
- R_BLE_GAP_StartAdv
- R_BLE_GAP_SetAdvParam
- R_BLE_GATT_GetMtu
- R_BLE_GATTS_SetDbInst
- R_BLE_GATTS_Init
- R_BLE_GATTS_GetAttr

- R_BLE_GATTS_SetAttr
- R_BLE_GATTS_Notification
- R_BLE_GATTS_Indication
- R_BLE_GATTS_RegisterCb
- R_BLE_GATTC_Init
- R_BLE_GATTC_RegisterCb
- R_BLE_VS_SetBdAddr
- R_BLE_VS_SetTxPower

Currently supported callback events:

- BLE_GAP_EVENT_CONN_IND
- BLE_GAP_EVENT_DISCONN_IND
- BLE_GAP_EVENT_ADV_ON
- BLE_GAP_EVENT_ADV_OFF
- BLE_GATTC_EVENT_CONN_IND
- BLE_GATTS_EVENT_CONN_IND
- BLE_GATTC_EVENT_DISCONN_IND
- BLE_GATTS_EVENT_DISCONN_IND
- BLE_GATTS_EVENT_WRITE_RSP_COMP

# Transmit Power

Specific Operational Use Conditions for RYZ012 BLE module:

**Please consult the laws and regulations of the region(s) in which the device will be used to verify the information below before using this device.**

North America (FCC) The module must not be operated at power levels above 10.0 dBm. Host devices that need higher output power may not be marketed without prior re-certification.

Europe (RED) The module must not be operated at power levels above 8.4 dBm. Host devices that need higher output power may not be marketed in regions covered by RED regulation without prior re-certification.

Japan (MIC) The module must be operated at power level 4.5 dBm. Host devices that need to change output power may not be marketed without prior re-certification.

The current default level used in the e² studio Configurator is 4.57 dBm for the transmit power level.

# Examples

### BLE_ABS_SPP Basic Example

This is a basic example of minimal use of the BLE_ABS_SPP in an application.

```
/* The callback is called when peripheral event occurs. */
void gap_peripheral_cb (uint16_t type, ble_status_t result, st_ble_evt_data_t *
p_data)
{
 FSP_PARAMETER_NOT_USED(result);
```

```
 FSP_PARAMETER_NOT_USED(p_data);
 switch (type)
    {
 case BLE_GAP_EVENT_STACK_ON:
     {
           g_ble_event_flag = g_ble_event_flag | BLE_ABS_EVENT_FLAG_STACK_ON;
 break;
      }
 case BLE_GAP_EVENT_ADV_ON:
     {
 st_ble_gap_adv_set_evt_t * p_gap_adv_set_evt_param = (st_ble_gap_adv_set_evt_t *)
p_data->p_param;
           g_advertising_handle = p_gap_adv_set_evt_param->adv_hdl;
           g_ble_event_flag    |= BLE_ABS_EVENT_FLAG_ADV_ON;
 break;
      }
 case BLE_GAP_EVENT_ADV_OFF:
     {
           g_ble_event_flag |= BLE_ABS_EVENT_FLAG_ADV_OFF;
 break;
      }
 case BLE_GAP_EVENT_CONN_IND:
     {
           g_ble_event_flag |= BLE_ABS_EVENT_FLAG_CONN_IND;
 break;
      }
     {
 /* Do nothing. */
 break;
        }
 default:
 break;
    }
}
```

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 'E', 'x', 'a', 'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 'T', 'E', 'S', 'T', '_', 'E', 'x', 'a',
'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    uint8_t advertising_data[] =
    {
 /* Flags */
       0x02,
       0x01,
       (0x1a),
 /* Shortened Local Name */
       0x08,
       0x08,
       BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
    };
 /* Scan Response Data */
    uint8_t scan_response_data[] =
    {
 /* Complete Local Name */
       0x0D,
       0x09,
       BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
    };
```

```
 ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
    {
       .p_peer_address             =
NULL,
       .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
       .slow_advertising_period    =
0x0000,
       .p_advertising_data         =
advertising_data,
       .advertising_data_length    = sizeof
(advertising_data),
       .p_scan_response_data       =
scan_response_data,
       .scan_response_data_length = sizeof
(scan_response_data),
       .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
       .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
       .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
       .own_bluetooth_address      = {0},
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure the Transmit Level */
    err = R_BLE_VS_SetTxPower(0, BLE_ABS_TRANSMIT_POWER);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_STACK_ON event is notified. */
```

```
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if (BLE_ABS_EVENT_FLAG_ADV_OFF & g_ble_event_flag)
      {
 /* Restart advertise, when stop advertising. */
      err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 if (FSP_SUCCESS == err)
      {
                g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
        }
 else if (FSP_ERR_INVALID_STATE == err)
      {
 /* BLE driver state is busy. */
                ;
        }
 else
      {
 /* Handle any errors. This function should be defined by the user. */
      assert(FSP_SUCCESS == err);
        }
       }
```

```
 else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
      {
 /* Stop advertising after a certain amount of time */
 R_BLE_GAP_StopAdv(g_advertising_handle);
       }
 else
      {
            ;
       }
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
 /* Clean up & Close BLE driver */
    g_ble_event_flag = 0;
 /* Close BLE driver */
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### BLE_ABS_SPP Firmware Update Example

This is a basic example of performing a firmware update.

```
void ble_abs_firmware_update_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
    assert(FSP_SUCCESS == err);
 /* Complete the firmware update (This function blocks until the procedure has
completed). */
    ble_status_t status = R_BLE_VS_UpdateModuleFirmware(g_firmware_image,
g_firmware_image_size);
    assert(BLE_SUCCESS == status);
```

```
 /* Close the module. */
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
    assert(FSP_SUCCESS == err);
 /* The RYZ012 is now using the updated firmware. Call open to re-initialize the
module using the new firmware. */
}
```

### BLE_ABS_SPP Firmware Update Asynchronous Example

This is an example of performing a firmware update using the asynchronous API.

```
volatile uint32_t g_firmware_update_complete = 0;
void vs_cb (uint16_t type, ble_status_t result, st_ble_vs_evt_data_t * p_data)
{
 FSP_PARAMETER_NOT_USED(p_data);
 if (BLE_SUCCESS != result)
    {
 /* Stop the firmware update process if there was an error. */
 return;
    }
 static uint16_t index = 0;
 switch (type)
    {
 /* The response to the START_FW_UPDATE command was received (Fall through to send
the next data frame). */
 case BLE_VS_EVENT_START_FW_UPDATE_COMP:
 /* The response to the SEND_FW_DATA command was received. */
 case BLE_VS_EVENT_SEND_FW_DATA_COMP:
     {
 if (BLE_VS_EVENT_START_FW_UPDATE_COMP == index)
     {
                index = 0;
        }
           uint16_t length          = 0;
           uint8_t * p_firmware_data = NULL;
```

```
 /*
  * Get next firmware data frame.
  * Note that app_data_frame_get() is an application defined function.
  */
            app_data_frame_get(index, &p_firmware_data, &length);
 if (length > 0)
      {
 /* Send the firmware data frame. */
 R_BLE_VS_SendFirmwareData(index, length, p_firmware_data);
                index++;
         }
 else
      {
 /* The previously completed data frame was the last data frame. Send the firmware
update end command. */
 R_BLE_VS_EndFirmwareUpdate(index - 1);
         }
 break;
       }
 /* The response to the END_FW_UPDATE command was received. */
 case BLE_VS_EVENT_END_FW_UPDATE_COMP:
      {
 /* After the BLE_VS_EVENT_END_FW_UPDATE_COMP command is sent, send the restart
command. */
 R_BLE_VS_RestartModule();
 break;
       }
 /* The module has finished restarting and the MODULE_READY event has been received.
*/
 case BLE_VS_EVENT_MODULE_READY_COMP:
      {
 /* Firmware update process has completed. */
            g_firmware_update_complete = 1;
 break;
```

```
      }
 default:
     {
 break;
      }
   }
}
void ble_abs_firmware_update_async_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
   assert(FSP_SUCCESS == err);
 /* Start the firmware update process. */
   ble_status_t status = R_BLE_VS_StartFirmwareUpdate();
   assert(BLE_SUCCESS == status);
 /* main loop */
 while (0 == g_firmware_update_complete)
   {
 /* Process BLE Event */
 R_BLE_Execute();
   }
 /* Close the module. */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
   assert(FSP_SUCCESS == err);
 /* The RYZ012 is now using the updated firmware. Call open to re-initialize the
module using the new firmware. */
}
```

### 5.2.12.31 Socket Communication Device (rm_comms_socket)

Modules » Networking

#### Functions

| | |
|---|---|
| fsp_err_t | RM_COMMS_SOCKET_Open (rm_comms_ctrl_t *const p_api_ctrl, |

| | |
|---|---|
| | rm_comms_cfg_t const *const p_cfg)<br>Opens and configures the SOCKET Comms module. Implements rm_comms_api_t::open. More... |
| fsp_err_t | RM_COMMS_SOCKET_Close (rm_comms_ctrl_t *const p_api_ctrl)<br>Disables specified SOCKET Comms module. Implements rm_comms_api_t::close. More... |
| fsp_err_t | RM_COMMS_SOCKET_CallbackSet (rm_comms_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)<br>Updates the SOCKET Comms callback. Implements rm_comms_api_t::callbackSet. More... |
| fsp_err_t | RM_COMMS_SOCKET_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)<br>Performs a read from the SOCKET bus. Implements rm_comms_api_t::read. More... |
| fsp_err_t | RM_COMMS_SOCKET_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)<br>Performs a write to the SOCKET bus. Implements rm_comms_api_t::write. More... |
| fsp_err_t | RM_COMMS_SOCKET_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params)<br>Performs simultaneous read and write. Implements rm_comms_api_t::writeRead.. More... |

### Detailed Description

Middleware to implement a generic communications interface over LWIP. This module implements the Communications Middleware Interface.

# Overview

The RM_COMMS_SOCKET module implements COMMS API for LWIP interface.

### Features

The implementation of the LWIP communications interfacehas the following key features:

- Non-blocking API for bare metal
- Non-blocking and blocking API for RTOS
- Socket connection detection mode

# Configuration

### Build Time Configurations for rm_comms_socket

The following build time configurations are defined in fsp_cfg/rm_comms_socket_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Networking > Socket Communication Device (rm_comms_socket)

This module can be added to the Stacks tab via New Stack > Networking > Socket Communication Device (rm_comms_socket).

| Configuration | Options | Default | Description |
|---|---|---|---|
| RTOS | | | |
| Recursive Write Mutex | • Do Not Use<br>• Use | Use | Lock device for writing in using RTOS. |
| Recursive Read Mutex | • Do Not Use<br>• Use | Use | Lock device for reading in using RTOS. |
| Mutex Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for recursive mutex operation in using RTOS. |
| Write Semaphore | • Do Not Use<br>• Use | Use | Block writing in using RTOS. |
| Read Semaphore | • Do Not Use<br>• Use | Use | Block reading in using RTOS. |
| Semaphore Timeout | Value must be a non-negative integer | 0xFFFFFFFF | Timeout for semaphore operation in using RTOS. |
| Name | Name must be a valid C symbol | g_comms_socket0 | Module name. |
| Socket type | TCP | TCP | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. |
| Server Port | Value must be a non-negative integer | 9010 | TCP port ranges from 1 to 63556 |

# Usage Notes

### Limitations

- If DHCP is enabled, user have to find board's IP address via DHCP server, mDNS, etc.
- UDP socket is not supported.
- RM_COMMS_SOCKET_WriteRead API is not implemented.

## Examples

### Basic Example

This is a basic example of minimal use of LWIP communications implementation in an application.

```
void rm_comms_socket_example () {
 /* This step: Initialize pin setting (of MCU device) */
 /* TODO code */
    rm_comms_socket_extended_cfg_t * p_extend = (rm_comms_socket_extended_cfg_t *)
g_comms_socket_port_cfg.p_extend;
 struct netif comms_socket_netif;
 /* Following code are lwIP setup process. */
    lwip_init();
 /* Initialize netif before open rm_comms_socket. */
    netif_add(&comms_socket_netif,
              RM_COMMS_SOCKET_EXAMPLE_IP_ADDR,
              RM_COMMS_SOCKET_EXAMPLE_NETMASK,
              RM_COMMS_SOCKET_EXAMPLE_GW,
              p_extend->p_lwip_ether,
 rm_lwip_ether_init,
              netif_input);
    netif_set_default(&comms_socket_netif);
    netif_set_up(&comms_socket_netif);
    netif_set_link_up(&comms_socket_netif);
 /* Init the rm_comms_socket */
 RM_COMMS_SOCKET_Open(&g_comms_socket_port_ctrl, &g_comms_socket_port_cfg);
 /* Waiting few milliseconds for socket ready to be used */
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 /* Set callback function */
 RM_COMMS_SOCKET_CallbackSet(&g_comms_socket_port_ctrl, tcp_callback,
g_comms_socket_port_cfg.p_context);
```

```
 /* This step: Client start to creating the socket and send connect command to Server
(MCU device) */
 /* TODO code */
 /* Waiting for connection to be established. */
    LWIP_TEST_WAIT_TIMEOUT(true == connected, LWIP_TEST_MAXIMUM_WAITING_TIME);
 /* Tries 3 times loop to receive/send data from/to client */
 for (uint16_t tries = 1; tries < 3; tries++)
    {
 /* This step: Server (MCU device) starts to receive data */
 RM_COMMS_SOCKET_Read(&g_comms_socket_port_ctrl, rx_tcp_data, BUFFER_SIZE);
 /* This step: Client starts to send the data to Server (MCU device) */
 /* TODO code */
 /* Wait for receiving data, RX complete flag will be indicated in callback function
*/
       LWIP_TEST_WAIT_TIMEOUT(true == rx_complete, LWIP_TEST_MAXIMUM_WAITING_TIME);
 /* Reset flag */
       rx_complete = false;
 /* Server (MCU device) starts sending package */
 RM_COMMS_SOCKET_Write(&g_comms_socket_port_ctrl, tx_tcp_data, BUFFER_SIZE);
 /* Wait for TX completed */
       LWIP_TEST_WAIT_TIMEOUT(true == tx_complete, LWIP_TEST_MAXIMUM_WAITING_TIME);
 /* This step: Client starts to receive data from server (MCU device) */
 /* TODO code */
 /* Clear the TX flag */
       tx_complete = false;
    }
 /* Close the socket server */
 RM_COMMS_SOCKET_Close(&g_comms_socket_port_ctrl);
}
void tcp_callback (rm_comms_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case RM_COMMS_EVENT_ERROR:
```

```
     {
           err = true;
break;
     }
case RM_COMMS_EVENT_RX_OPERATION_COMPLETE:
     {
           rx_complete = true;
break;
     }
case RM_COMMS_EVENT_TX_OPERATION_COMPLETE:
     {
           tx_complete = true;
break;
     }
case RM_COMMS_EVENT_OPERATION_COMPLETE:
     {
           connected = true;
break;
     }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_comms_socket_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_comms_socket_instance_ctrl_t

| struct rm_comms_socket_instance_ctrl_t | |
|---|---|
| Communications middleware control structure. | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_comms_cfg_t const * | p_cfg |

| | |
|---|---|
| | Middleware configuration. |
| | |
| rm_comms_socket_extended_cfg_t * | p_extend |
| | Pointer to device specific settings (lower level configuration structure) |
| | |
| void(* | p_callback )(rm_comms_callback_args_t *p_args) |
| | Pointer to callback that is called when a socket_event_t occurs. |
| | |
| void * | p_context |
| | Pointer to context passed into callback function. |
| | |

### Function Documentation

#### ◆ RM_COMMS_SOCKET_Open()

fsp_err_t RM_COMMS_SOCKET_Open ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg* )

Opens and configures the SOCKET Comms module. Implements rm_comms_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SOCKET Comms module successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ABORTED | Module is not successfully open. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_UNSUPPORTED | Socket type is not supported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SOCKET_Close()

fsp_err_t RM_COMMS_SOCKET_Close ( rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified SOCKET Comms module. Implements rm_comms_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_UNSUPPORTED | Socket type is not supported. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SOCKET_CallbackSet()

fsp_err_t RM_COMMS_SOCKET_CallbackSet ( rm_comms_ctrl_t *const *p_api_ctrl*, void(*)(rm_comms_callback_args_t *) *p_callback*, void *const *p_context* )

Updates the SOCKET Comms callback. Implements rm_comms_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_COMMS_SOCKET_Read()

fsp_err_t RM_COMMS_SOCKET_Read ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Performs a read from the SOCKET bus. Implements rm_comms_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_ENABLED | Operation is disabled when there is no TCP connection. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

*Do not call this API in Interrupt function when enabling mutex or semaphore*

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SOCKET_Write()

fsp_err_t RM_COMMS_SOCKET_Write ( rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes* )

Performs a write to the SOCKET bus. Implements rm_comms_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully writing data . |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_WRITE_FAILED | No bytes were transferred. |
| FSP_ERR_NOT_ENABLED | Operation is disabled when there is no active TCP connection. |
| FSP_ERR_NOT_OPEN | Module is not open. |

***Note***

*Do not call this API in Interrupt function when enabling mutex or semaphore*

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_COMMS_SOCKET_WriteRead()

fsp_err_t RM_COMMS_SOCKET_WriteRead ( rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params* )

Performs simultaneous read and write. Implements rm_comms_api_t::writeRead..

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer was started successfully. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Read bytes and write bytes are not the same. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.12.32 WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | RM_WIFI_DA16XXX_SntpServerIpAddressSet (uint8_t *p_server_ip_addr) |
| fsp_err_t | RM_WIFI_DA16XXX_SntpEnableSet (wifi_da16xxx_sntp_enable_t enable) |
| fsp_err_t | RM_WIFI_DA16XXX_SntpTimeZoneSet (int utc_offset_in_hours, uint32_t minutes, wifi_da16xxx_sntp_daylight_savings_enable_t daylightSavingsEnable) |
| fsp_err_t | RM_WIFI_DA16XXX_LocalTimeGet (uint8_t *p_local_time, uint32_t size_string) |
| fsp_err_t | RM_WIFI_DA16XXX_GenericAtSendRcv (char const *const at_string, char *const response_buffer, uint32_t length) |

### Detailed Description

Wifi and Socket implementation using the DA16XXX WiFi module on RA MCUs.

# Overview

This Middleware module currently supplies an implementation for using the DA16XXX module in two different configurations.

The first is for the FreeRTOS WiFi interface, where the network stack is located on the MCU using a socket interface, and the DA16XXX Wi-Fi module provides a connection to the internet. This configuration requires FreeRTOS to be used.

The second configuration is for use with the DA16XXX on-chip MQTT and HTTP client modules, where the network stack is located on the DA16XXX module. In this configuration, this Middleware driver supports FreeRTOS as well as Baremetal (Microsoft Azure support expected in future release).

DA16XXX is a low power Wi-Fi networking SoC that delivers a dramatic breakthrough in battery life even for devices that are continuously connected to the Wi-Fi network. The module comes readily equipped with radio certification for Japan, North America and Europe. More information about this module can be found at the DA16XXX Web Site

The DA16XXX module supports the following modules:

- DA16200
- DA16600 (currently Wi-Fi features only)

**Features**

The WiFi Onchip da16xxx Middleware driver supplies these features:

- Supports connect/disconnect to a b/g/n (2.4GHz) WiFi Access Point using Open, WPA, and WPA2 security. Encryption types can be either TKIP, or CCMP(AES).
- Supports retrieval of the module device MAC address.
- Once connected you can acquire the assigned module device IP.
- Supports a WiFi network scan capability to get a list of local Access Points.
- Supports a Ping function to test network connectivity.
- Supports a DNS Query call to retrieve the IPv4 address of a supplied URL.
- Supports TCP client sockets.
- Drive supports 1 UART for interfacing with the DA16XXX module.

# Configuration

### Build Time Configurations for rm_wifi_da16xxx

The following build time configurations are defined in fsp_cfg/rm_wifi_da16xxx_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Enable DA16600 Support | • Enabled<br>• Disabled | Disabled | Enabled if using the DA16600 module |
| Check Minimum SDK Version | • Disabled<br>• Enabled | Enabled | Enable or disable comparing the current DA16XXX module's SDK version to the minimum required |

| | | | |
|---|---|---|---|
| Number of supported socket instances (not used with on-chip APIs) | 1 | 1 | Enable number of socket instances |
| Size of RX buffer for socket (not used with on-chip APIs) | Manual Entry | 8192 | |
| Size of TX buffer for CMD Port | Manual Entry | 1500 | |
| Size of RX buffer for CMD Port | Manual Entry | 3000 | |
| Enable SNTP Client | • Enabled<br>• Disabled | Disabled | Should the SNTP client of the module be enabled |

**Configurations for Networking > WiFi DA16XXX Framework Driver (rm_wifi_da16xxx)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Country code in ISO 3166-1 alpha-2 standard | Manual Entry | US | |
| SNTP server IPv4 address | Must be a valid IPv4 address | 0.0.0.0 | |
| SNTP Timezone UTC Offset in Hours | Refer to the RA Configuration tool for available options. | 0 | Value in hours from 12 to -12 |

Note: It is suggested that when using the DA16XXX module that DTC and FIFO are enabled in the UART configuration to facilitate a more reliable data transfer between module and MCU.

Note: If you wish to use flow control then you must enable flow control in the RA Configuration editor. This can be found in the UART setting. It is advantageous to use flow control all the time since it allows the hardware to gate the flow of data across the serial bus. Without hardware flow control for faster data rate you will most likely see an overflow condition between MCU and the module device.

Note: Higher baud rates are supported in the RA Configuration editor and should be changed in the first UART configuration. There is no need to change the second UART baud rate since it is only used as an AT command channel.

Note: It is a good idea to also enable the FIFO in the UART configuration settings if you plan to use higher baud rates.

### Interrupt Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx).

### Clock Configuration

Refer to UART (r_sci_uart).

### Pin Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip DA16XXX Framework Driver (rm_wifi_da16xxx)

# Usage Notes

- When the DA16XXX Wi-Fi configures the UART baud rate on the DA16XXX module based on the user settings in the UART driver configuration properties, the UART Baud rate settings will be stored in the NVM memory of the DA16XXX. After reboot, the DA16XXX comes up with the previously updated UART baud rate settings.
- When using the DA16600 Pmod module, make sure that the SDK has been updated to the latest version that supports AT commands. The default SDK that comes from the factory doesn't support the AT command interface, which will cause the WIFI_On() API to fail. This is not required for the DA16200 Pmod module.

### Limitations

- WiFi AP connections do not currently support WEP security.
- The default UART baud rate supported by v3.2.1 Wi-Fi SDK is 115200 and v3.2.4 Wi-Fi SDK is 230400. User needs to explicitly configure the default UART baud settings in the UART driver configurator properties based on the version of Wi-Fi SDK used in their testing.
- This driver is configured to support the following baud rates only: 115200, 230400, 460800, and 921600. These values are defined by the uart_baud_rates[] array and UART_BAUD_MAX_CNT found in the transport layer files.
- In v3.2.1 Wi-Fi SDK, the daylight savings time setting is disabled by default. The user needs to mandatorily set the following parameters such as minutes = 0, daylight savings to disable when calling RM_WIFI_DA16XXX_SntpTimeZoneSet() API.
- Network connection parameters SSID and Passphrase for the Access Point can not contain any commas. This is a current limitation of the da16xxx module firmware. The rm_wifi_da16xxx_connect() function will return an error if a comma is detected.
- Wi-Fi AP Scanning is currently limited to max of 10 Access Points.

# Examples

### Basic Example

This is a basic example of minimal use of WiFi Middleware in an application.

```
void wifi_basic_example (void)
{
 WIFIReturnCode_t wifi_err;
 /* Setup Access Point connection parameters */
 WIFINetworkParams_t net_params =
    {
       .ucChannel                  = 0,
       .xPassword.xWPA.cPassphrase = "password",
       .ucSSID                     = "access_point_ssid",
```

```
      .xPassword.xWPA.ucLength = 8,
      .ucSSIDLength            = 17,
      .xSecurity               = eWiFiSecurityWPA2,
   };
   memset(scan_data, 0, sizeof(WIFIScanResult_t) * MAX_WIFI_SCAN_RESULTS);
   memset(g_socket_recv_buffer, 0, sizeof(uint8_t) * SX_WIFI_SOCKET_RX_BUFFER_SIZE);
/* Open connection to the Wifi Module */
   wifi_err = WIFI_On();
   assert(eWiFiSuccess == wifi_err);
/* Connect to the Access Point */
   wifi_err = WIFI_ConnectAP(&net_params);
   assert(eWiFiSuccess == wifi_err);
/* Get address assigned by AP */
WIFIIPConfiguration_t ipInfo;
   wifi_err = WIFI_GetIPInfo(&ipInfo);
   assert(eWiFiSuccess == wifi_err);
/* Ping an address accessible on the network */
   uint8_t       ip_address[4] = {216, 58, 194, 174}; // NOLINT
const uint16_t ping_count    = 3;
const uint32_t intervalMS    = 100;
   wifi_err = WIFI_Ping(&ip_address[0], ping_count, intervalMS);
   assert(eWiFiSuccess == wifi_err);
/* Scan the local Wifi network for other APs */
   wifi_err = WIFI_Scan(&scan_data[0], MAX_WIFI_SCAN_RESULTS);
   assert(eWiFiSuccess == wifi_err);
/* Shutdown the WIFI */
WIFI_Off();
}
```

### SNTP example

An example of using Simple Network Time Protocol (SNTP) on WiFi in an application.

```
#define RM_WIFI_DA16XXX_TEMP_BUFFER_SIZE (64)
/*
```

```
 * Example of the use of SNTP with Wifi. Example gets the epoch time and local
 * system time strings. It is also demonstrated how the user will need to disconnect
 * from the access point to make changes to the SNTP configuration during runtime.
 *
 * Function assumes that the SNTP has been enabled and configured with proper
 * SNTP server address. For brevity error checking has not been implemented.
 *
 */
void wifi_sntp_example (void)
{
 /* Setup Access Point connection parameters */
 WIFINetworkParams_t net_params =
    {
       .ucSSID                    = "access_point_ssid",
       .ucSSIDLength              = 17,
       .xPassword.xWPA.cPassphrase = "password",
       .xPassword.xWPA.ucLength    = 8,
       .ucChannel                 = 0,
       .xSecurity                 = eWiFiSecurityWPA2
    };
    uint8_t local_time[RM_WIFI_DA16XXX_TEMP_BUFFER_SIZE];
    time_t current_sys_time = 0;
 // SNTP IP address
    uint8_t ip_address_sntp_server_valid[4] = {216, 239, 35, 0}; // NOLINT : Static
IP address
    memset(local_time, 0, sizeof(local_time));
 /* Open connection to the Wifi Module */
 WIFI_On();
 /* Connect to the Access Point */
 WIFI_ConnectAP(&net_params);
 /* Get the local time string */
 RM_WIFI_DA16XXX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the access point to make changes to the SNTP configuration */
 WIFI_Disconnect();
```

```
 /* Change the IP address of the server */
 RM_WIFI_DA16XXX_SntpServerIpAddressSet((uint8_t *) ip_address_sntp_server_valid);
 /* Change the timezone to PST with daylight saving enabled */
 RM_WIFI_DA16XXX_SntpTimeZoneSet(-8, 0, WIFI_DA16XXX_SNTP_DAYLIGHT_SAVINGS_DISABLE);
 /* Connect back to the access point */
 WIFI_ConnectAP(&net_params);
 /* Get the local time string in format [DayOfWeek Month DayOfMonth Year
Hour:Minute:Second] */
 RM_WIFI_DA16XXX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the Access Point and shutdown the WIFI module*/
 WIFI_Disconnect();
 WIFI_Off();
}
```

## Data Structures

| | |
|---|---|
| struct | wifi_da16xxx_cfg_t |
| struct | da16xxx_socket_t |
| struct | wifi_da16xxx_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | wifi_da16xxx_sntp_enable_t |
| enum | da16xxx_socket_status_t |
| enum | da16xxx_socket_rw |
| enum | da16xxx_recv_state |
| enum | wifi_da16xxx_sntp_daylight_savings_enable_t |

## Data Structure Documentation

### ◆ wifi_da16xxx_cfg_t

| struct wifi_da16xxx_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| at_transport_da16xxx_instance_t | p_transport_instance | |

| | | |
|---|---|---|
| const * | | |
| const uint32_t | num_sockets | Number of sockets to initialize. |
| const uint8_t * | country_code | Country code defined in ISO3166-1 alpha-2 standard. |
| const uint8_t * | sntp_server_ip | The SNTP server IP address string. |
| const int32_t | sntp_utc_offset_in_hours | Timezone offset in secs (-43200 - 43200) |
| void * | p_context | User defined context passed into callback function. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ da16xxx_socket_t

| struct da16xxx_socket_t | | |
|---|---|---|
| DA16XXX Wifi internal socket instance structure | | |
| Data Fields | | |
| uint8_t | remote_ipaddr[4] | Remote IP address. |
| int | remote_port | Remote Port. |
| int | socket_recv_data_len | Data length of incoming socket data. |
| int | socket_type | Socket type (TCP Server \| TCP Client \| UDP) |
| uint32_t | socket_status | Current socket status. |
| uint32_t | socket_recv_error_count | Socket receive error count. |
| uint32_t | socket_create_flag | Flag to determine in socket has been created. |
| uint32_t | socket_read_write_flag | flag to determine if read and/or write channels are active. |
| da16xxx_recv_state | socket_recv_state | Incoming Socket data header information. |
| StreamBufferHandle_t | socket_byteq_hdl | Socket stream buffer handle. |
| StaticStreamBuffer_t | socket_byteq_struct | Structure to hold stream buffer info. |
| uint8_t | socket_recv_buff[WIFI_DA16XXX_CFG_MAX_SOCKET_RX_SIZE] | Socket receive buffer used by byte queue. |

### ◆ wifi_da16xxx_instance_ctrl_t

| struct wifi_da16xxx_instance_ctrl_t |
|---|
| |

| WIFI_DA16XXX private control block. DO NOT MODIFY. | | |
|---|---|---|
| Data Fields | | |
| wifi_da16xxx_cfg_t const * | p_wifi_da16xxx_cfg | Pointer to initial configurations. |
| uint32_t | num_creatable_sockets | Number of simultaneous sockets supported. |
| uint32_t | curr_socket | Current Socket index for AT commands. |
| uint32_t | open | Flag to indicate if wifi instance has been initialized. |
| uint8_t | is_sntp_enabled | Flag to indicate Enable/Disable of SNTP Client. |
| uint8_t | cmd_tx_buff[WIFI_DA16XXX_CFG_CMD_TX_BUF_SIZE] | Command send buffer. |
| uint8_t | cmd_rx_buff[WIFI_DA16XXX_CFG_CMD_RX_BUF_SIZE] | Command receive buffer. |
| volatile uint32_t | curr_socket_index | Currently active socket instance. |
| uint8_t | curr_ipaddr[4] | Current IP address of module. |
| uint8_t | curr_subnetmask[4] | Current Subnet Mask of module. |
| uint8_t | curr_gateway[4] | Current GAteway of module. |
| da16xxx_socket_t | sockets[WIFI_DA16XXX_CFG_NUM_CREATEABLE_SOCKETS] | Internal socket instances. |

## Enumeration Type Documentation

### ◆ wifi_da16xxx_sntp_enable_t

| enum wifi_da16xxx_sntp_enable_t |
|---|
| DA16XXX WiFi module enable/disable for SNTP |

### ◆ da16xxx_socket_status_t

| enum da16xxx_socket_status_t |
|---|
| DA16XXX Wifi socket status types |

### ◆ da16xxx_socket_rw

| enum da16xxx_socket_rw |
|---|
| DA16XXX socket shutdown channels |

### ◆ da16xxx_recv_state

| enum da16xxx_recv_state |
| --- |
| DA16XXX socket receive state |

### ◆ wifi_da16xxx_sntp_daylight_savings_enable_t

| enum wifi_da16xxx_sntp_daylight_savings_enable_t |
| --- |
| DA16XXX WiFi module enable/disable for SNTP Daylight |

## Function Documentation

### ◆ RM_WIFI_DA16XXX_SntpServerIpAddressSet()

fsp_err_t RM_WIFI_DA16XXX_SntpServerIpAddressSet ( uint8_t * *p_server_ip_addr*)

Set the SNTP Client Server IP Address

**Parameters**

| | | |
| --- | --- | --- |
| [in] | p_server_ip_addr | Pointer to IP address of SNTP server in byte array format. |

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_ASSERTION | The parameter p_server_ip_addr is NULL. |

Update the SNTP Server IP Address

### ◆ RM_WIFI_DA16XXX_SntpEnableSet()

fsp_err_t RM_WIFI_DA16XXX_SntpEnableSet ( wifi_da16xxx_sntp_enable_t *enable*)

Set the SNTP Client to Enable or Disable

**Parameters**

| | | |
|---|---|---|
| [in] | enable | Flag to indicate enable/disable for SNTP support. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

Enable or Disable the SNTP Client Service

### ◆ RM_WIFI_DA16XXX_SntpTimeZoneSet()

fsp_err_t RM_WIFI_DA16XXX_SntpTimeZoneSet ( int *utc_offset_in_hours*, uint32_t *minutes*, wifi_da16xxx_sntp_daylight_savings_enable_t *daylightSavingsEnable* )

Set the SNTP Client Timezone

**Parameters**

| | | |
|---|---|---|
| [in] | utc_offset_in_hours | Timezone in UTC offset in hours |
| [in] | minutes | Number of minutes used for timezone offset from GMT. |
| [in] | daylightSavingsEnable | Enable/Disable daylight saving in the timezone calculation. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Parameter passed into function was invalid. |

Update the SNTP Timezone

### ◆ RM_WIFI_DA16XXX_LocalTimeGet()

fsp_err_t RM_WIFI_DA16XXX_LocalTimeGet ( uint8_t * *p_local_time*, uint32_t *size_string* )

Get the current local time based on current timezone in a string . Exp: YYYY-MM-DD,HOUR:MIN:SECS

**Parameters**

| | | |
|---|---|---|
| [out] | p_local_time | Returns local time in string format. |
| [in] | size_string | Size of p_local_time string buffer.The size of this string needs to be at least 25 bytes |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully returned the local time string. |
| FSP_ERR_ASSERTION | The parameter local_time or p_instance_ctrl is NULL. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | String size value passed in exceeds maximum. |

Get the current local time based on current timezone in a string format

### ◆ RM_WIFI_DA16XXX_GenericAtSendRcv()

fsp_err_t RM_WIFI_DA16XXX_GenericAtSendRcv ( char const *const *at_string*, char *const *response_buffer*, uint32_t *length* )

Sends any AT command compatible with the DA16XXX module. Provide optional buffer to receive the response.

**Parameters**

| | | |
|---|---|---|
| [in] | at_string | Input AT command string from the user. |
| [in] | response_buffer | Optional buffer for receiving the response. |
| [in] | length | Size of optional buffer. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Input was not a valid AT command. |

Send a generic AT command to the DA16XXX and optionally receive a response

## 5.2.12.33 WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex)

Modules » Networking

### Functions

| | |
|---|---|
| fsp_err_t | rm_wifi_onchip_silex_open (wifi_onchip_silex_cfg_t const *const p_cfg) |
| fsp_err_t | rm_wifi_onchip_silex_close () |
| fsp_err_t | rm_wifi_onchip_silex_disconnect () |
| fsp_err_t | rm_wifi_onchip_silex_socket_connected (fsp_err_t *p_status) |
| fsp_err_t | rm_wifi_onchip_silex_network_info_get (uint32_t *p_ip_addr, uint32_t *p_subnet_mask, uint32_t *p_gateway) |
| fsp_err_t | rm_wifi_onchip_silex_connect (const char *p_ssid, WIFISecurity_t security, const char *p_passphrase) |

| | |
|---|---|
| fsp_err_t | rm_wifi_onchip_silex_mac_addr_get (uint8_t *p_macaddr) |
| fsp_err_t | rm_wifi_onchip_silex_scan (WIFIScanResult_t *p_results, uint32_t maxNetworks) |
| fsp_err_t | rm_wifi_onchip_silex_ping (uint8_t *p_ip_addr, uint32_t count, uint32_t interval_ms) |
| fsp_err_t | rm_wifi_onchip_silex_ip_addr_get (uint32_t *p_ip_addr) |
| fsp_err_t | rm_wifi_onchip_silex_avail_socket_get (uint32_t *p_socket_id) |
| fsp_err_t | rm_wifi_onchip_silex_socket_status_get (uint32_t socket_no, uint32_t *p_socket_status) |
| int32_t | rm_wifi_onchip_silex_tcp_shutdown (uint32_t socket_no, uint32_t shutdown_channels) |
| fsp_err_t | rm_wifi_onchip_silex_socket_create (uint32_t socket_no, uint32_t type, uint32_t ipversion) |
| fsp_err_t | rm_wifi_onchip_silex_tcp_connect (uint32_t socket_no, uint32_t ipaddr, uint32_t port) |
| fsp_err_t | rm_wifi_onchip_silex_udp_connect (uint32_t socket_no, uint32_t ipaddr, uint32_t port, uint32_t type) |
| int32_t | rm_wifi_onchip_silex_send (uint32_t socket_no, const uint8_t *p_data, uint32_t length, uint32_t timeout_ms) |
| int32_t | rm_wifi_onchip_silex_recv (uint32_t socket_no, uint8_t *p_data, uint32_t length, uint32_t timeout_ms) |
| fsp_err_t | rm_wifi_onchip_silex_socket_disconnect (uint32_t socket_no) |
| fsp_err_t | rm_wifi_onchip_silex_dns_query (const char *p_textstring, uint8_t *p_ip_addr) |
| fsp_err_t | RM_WIFI_ONCHIP_SILEX_EpochTimeGet (time_t *p_utc_time) |
| fsp_err_t | RM_WIFI_ONCHIP_SILEX_LocalTimeGet (uint8_t *p_local_time, uint32_t size_string) |
| fsp_err_t | RM_WIFI_ONCHIP_SILEX_SntpEnableSet (wifi_onchip_silex_sntp_enable_t enable) |
| fsp_err_t | RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet (uint8_t *p_ip_address) |
| fsp_err_t | RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet (int32_t hours, uint32_t |

| | |
|---|---|
| | minutes, wifi_onchip_silex_sntp_daylight_savings_enable_t daylightSavingsEnable) |

### Detailed Description

Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs.

# Overview

This Middleware module supplies an implementation for the FreeRTOS WiFi interface and AzureRTOS NetxDuo WiFi interface using the Silex SX-ULPGN module.

You can find specifics about the WiFi and Secure Socket interface APIs supported by this module at these web sites: Wifi API.

The SX-ULPGN is a low-power, compact IEEE 802.11b/g/n 2.4GHz 1x1 Wireless LAN module equipped with the Qualcomm® QCA4010 Wireless SOC. The module comes readily equipped with radio certification for Japan, North America and Europe. More information about this module can be found at the Silex Web Site

### Features

The WiFi Onchip Silex Middleware driver supplies these features:

- Supports connect/disconnect to a b/g/n (2.4GHz) WiFi Access Point using Open, WPA, and WPA2 security. Encryption types can be either TKIP, or CCMP(AES).
- Supports retrieval of the module device MAC address.
- Once connected you can acquire the assigned module device IP.
- Supports a WiFi network scan capability to get a list of local Access Points.
- Supports a Ping function to test network connectivity.
- Supports a DNS Query call to retrieve the IPv4 address of a supplied URL.
- Supports both TCP and UDP client sockets.
- Drive supports 1 or 2 UARTs for interfacing with the SX-ULPGN module. The second UART is considered optional.

# Configuration

### Build Time Configurations for rm_wifi_onchip_silex

The following build time configurations are defined in fsp_cfg/rm_wifi_onchip_silex_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Number of supported socket instances | Refer to the RA Configuration tool for available options. | 1 | Enable number of socket instances |
| Size of RX buffer for socket | Manual Entry | 4096 | |

| | | | |
|---|---|---|---|
| Size of TX buffer for CMD Port | Manual Entry | 1500 | |
| Size of RX buffer for CMD Port | Manual Entry | 3000 | |
| Semaphore maximum timeout | Manual Entry | 10000 | |
| Number of retries for AT commands | Manual Entry | 10 | |
| Module Reset Port | Refer to the RA Configuration tool for available options. | 06 | Specify the module reset pin port for the MCU. |
| Module Reset Pin | Refer to the RA Configuration tool for available options. | 03 | Specify the module reset pin for the MCU. |
| Enable SNTP Client | • Enabled<br>• Disabled | Disabled | Should the SNTP client of the module be enabled |

**Configurations for Networking > WiFi Onchip Silex Driver using UART (rm_wifi_onchip_silex)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| SNTP server IPv4 address | Must be a valid IPv4 address | 0.0.0.0 | |
| SNTP Timezone Offset from UTC Hours | Must be between 12 and -12 hours | 0 | Value in hours from 12 to -12 |
| SNTP Timezone Offset from UTC Minutes | Must be between 0 and 59 minutes | 0 | Value in minutes from 0 to 59 |
| Use Daylight Savings Time | • Enabled<br>• Disabled | Disabled | Specify if daytime savings should be used for local time calulation |

Note: It is suggested that when using the Silex Module that DTC and FIFO are enabled in the UART configuration to facilitate a more reliable data transfer between module and MCU.

Note: If you wish to use flow control then you must enable flow control in the RA Configuration editor. This can be found in the UART setting. It is advantageous to use flow control all the time since it allows the hardware to gate the flow of data across the serial bus. Without hardware flow control for faster data rate you will most likely see an overflow condition between MCU and the module device.

Note: Higher baud rates are supported in the RA Configuration editor and should be changed in the first UART configuration. There is no need to change the second UART baud rate since it is only used as an AT command channel.

Note: It is a good idea to also enable the FIFO in the UART configuration settings if you plan to use higher baud rates.

### Interrupt Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex).

### Clock Configuration

Refer to UART (r_sci_uart).

### Pin Configuration

Refer to UART (r_sci_uart). R_SCI_UART_Open() is called by WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex)

## Usage Notes

### Limitations

- WiFi AP connections do not currently support WEP security.
- When operating with a single UART only single socket connections are possible. To support multiple sockets two UART channels must be connected to the module. When using the Renesas-provided SX-ULPGN PMOD board the second UART channel is on pins 9 and 10 of the PMOD header.
- Network connection parameters SSID and Passphrase for the Access Point can not contain any commas. This is a current limitation of the Silex module firmware. The rm_wifi_onchip_silex_connect() function will return an error if a comma is detected.
- When operating with a single UART and there is an active socket connection you cannot call WIFI_Scan(), WIFI_Ping(), WIFI_GetMAC(), or WIFI_GetIPInfo(). Calling one of these function will return an error code in this situation. These commands are blocked in the one UART case during an active socket connection because they could cause data loss. To avoid this limitation please configure the hardware to use both UARTs.
- The Silex WiFi modules SNTP support requires all configuration changes to made when the WiFi is disconnected from an Access Point. This is a limitation of the Silex module firmware. If changes to the default SNTP settings are required then the application will have to close the current AP connection, make the necessary SNTP changes, and then re-establish the original connection.

## Examples

### Basic Example

This is a basic example of minimal use of WiFi Middleware in an application.

```
void wifi_onchip_basic_example (void)
{
 WIFIReturnCode_t wifi_err;
 /* Setup Access Point connection parameters */
 WIFINetworkParams_t net_params =
    {
       .ucChannel                  = 0,
```

```
      .xPassword.xWPA.cPassphrase = "password",
      .ucSSID                     = "access_point_ssid",
      .xPassword.xWPA.ucLength = 8,
      .ucSSIDLength            = 17,
      .xSecurity               = eWiFiSecurityWPA2,
   };
   memset(scan_data, 0, sizeof(WIFIScanResult_t) * MAX_WIFI_SCAN_RESULTS);
   memset(g_socket_recv_buffer, 0, sizeof(uint8_t) * SX_WIFI_SOCKET_RX_BUFFER_SIZE);
/* Open connection to the Wifi Module */
   wifi_err = WIFI_On();
   assert(eWiFiSuccess == wifi_err);
/* Connect to the Access Point */
   wifi_err = WIFI_ConnectAP(&net_params);
   assert(eWiFiSuccess == wifi_err);
/* Get address assigned by AP */
WIFIIPConfiguration_t ipInfo;
   wifi_err = WIFI_GetIPInfo(&ipInfo);
   assert(eWiFiSuccess == wifi_err);
/* Ping an address accessible on the network */
   uint8_t       ip_address[4] = {216, 58, 194, 174}; // NOLINT
const uint16_t ping_count    = 3;
const uint32_t intervalMS    = 100;
   wifi_err = WIFI_Ping(&ip_address[0], ping_count, intervalMS);
   assert(eWiFiSuccess == wifi_err);
/* Scan the local Wifi network for other APs */
   wifi_err = WIFI_Scan(&scan_data[0], MAX_WIFI_SCAN_RESULTS);
   assert(eWiFiSuccess == wifi_err);
/* Shutdown the WIFI */
WIFI_Off();
}
```

### SNTP example

An example of using Simple Network Time Protocol (SNTP) on WiFi in an application.

```
#define RM_WIFI_ONCHIP_SILEX_TEMP_BUFFER_SIZE (64)
/*
 * Example of the use of SNTP with Wifi. Example gets the epoch time and local
 * system time strings. It is also demonstrated how the user will need to disconnect
 * from the access point to make changes to the SNTP configuration during runtime.
 *
 * Function assumes that the SNTP has been enabled and configured with proper
 * SNTP server address. For brevity error checking has not been implemented.
 *
 */
void wifi_onchip_sntp_example (void)
{
 /* Setup Access Point connection parameters */
 WIFINetworkParams_t net_params =
    {
       .ucSSID                    = "access_point_ssid",
       .ucSSIDLength              = 17,
       .xPassword.xWPA.cPassphrase = "password",
       .xPassword.xWPA.ucLength    = 8,
       .ucChannel                 = 0,
       .xSecurity                 = eWiFiSecurityWPA2
    };
    uint8_t local_time[RM_WIFI_ONCHIP_SILEX_TEMP_BUFFER_SIZE];
    time_t current_sys_time = 0;
 // SNTP IP address
    uint8_t ip_address_sntp_server_valid[4] = {216, 239, 35, 0}; // NOLINT : Static
IP address
    memset(local_time, 0, sizeof(local_time));
 /* Open connection to the Wifi Module */
 WIFI_On();
 /* Connect to the Access Point */
 WIFI_ConnectAP(&net_params);
 /* Get the Epoch time in seconds since Jan 1, 1970 UTC */
 RM_WIFI_ONCHIP_SILEX_EpochTimeGet(&current_sys_time);
```

```
 /* Get the local time string */
 RM_WIFI_ONCHIP_SILEX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the access point to make changes to the SNTP configuration */
 WIFI_Disconnect();
 /* Change the IP address of the server */
 RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet((uint8_t *)
ip_address_sntp_server_valid);
 /* Change the timezone to PST with daylight saving enabled */
 RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet(-7, 0,
WIFI_ONCHIP_SILEX_SNTP_DAYLIGHT_SAVINGS_ENABLE);
 /* Connect back to the access point */
 WIFI_ConnectAP(&net_params);
 /* Get the Epoch time in seconds since Jan 1, 1970 UTC */
 RM_WIFI_ONCHIP_SILEX_EpochTimeGet(&current_sys_time);
 /* Get the local time string in format [DayOfWeek Month DayOfMonth Year
Hour:Minute:Second] */
 RM_WIFI_ONCHIP_SILEX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the Access Point and shutdown the WIFI module*/
 WIFI_Disconnect();
 WIFI_Off();
}
```

## Data Structures

| | |
|---|---|
| struct | wifi_onchip_silex_cfg_t |
| struct | ulpgn_socket_t |
| struct | wifi_onchip_silex_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | sx_ulpgn_socket_status_t |
| enum | sx_ulpgn_socket_rw |
| enum | wifi_onchip_silex_sntp_enable_t |
| enum | wifi_onchip_silex_sntp_daylight_savings_enable_t |

## Data Structure Documentation

### ◆ wifi_onchip_silex_cfg_t

| struct wifi_onchip_silex_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| const uint32_t | num_uarts | Number of UART interfaces to use. |
| const uint32_t | num_sockets | Number of sockets to initialize. |
| const bsp_io_port_pin_t | reset_pin | Reset pin used for module. |
| const uart_instance_t * | uart_instances[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | SCI UART instances. |
| const wifi_onchip_silex_sntp_enable_t | sntp_enabled | Enable/Disable the SNTP Client. |
| const uint8_t * | sntp_server_ip | The SNTP server IP address string. |
| const int32_t | sntp_timezone_offset_from_utc_hours | Timezone offset from UTC in (+/-) hours. |
| const uint32_t | sntp_timezone_offset_from_utc_minutes | Timezone offset from UTC in minutes. |
| const wifi_onchip_silex_sntp_daylight_savings_enable_t | sntp_timezone_use_daylight_savings | Enable/Disable use of daylight saving time. |
| void * | p_context | User defined context passed into callback function. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ ulpgn_socket_t

| struct ulpgn_socket_t | | |
|---|---|---|
| Silex ULPGN Wifi internal socket instance structure | | |
| Data Fields | | |
| StreamBufferHandle_t | socket_byteq_hdl | Socket stream buffer handle. |
| StaticStreamBuffer_t | socket_byteq_struct | Structure to hold stream buffer info. |
| uint8_t | socket_recv_buff[WIFI_ONCHIP_SILEX_CFG_MAX_SOCKET_RX_SIZE] | Socket receive buffer used by byte queue. |
| uint32_t | socket_status | Current socket status. |

| | | |
|---|---|---|
| uint32_t | socket_recv_error_count | Socket receive error count. |
| uint32_t | socket_create_flag | Flag to determine in socket has been created. |
| uint32_t | socket_read_write_flag | flag to determine if read and/or write channels are active. |

### ◆ wifi_onchip_silex_instance_ctrl_t

| struct wifi_onchip_silex_instance_ctrl_t | | |
|---|---|---|
| WIFI_ONCHIP_SILEX private control block. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | open | Flag to indicate if wifi instance has been initialized. |
| wifi_onchip_silex_cfg_t const * | p_wifi_onchip_silex_cfg | Pointer to initial configurations. |
| bsp_io_port_pin_t | reset_pin | Wifi module reset pin. |
| uint32_t | num_uarts | number of UARTS currently used for communication with module |
| uint32_t | tx_data_size | Size of the data to send. |
| uint32_t | num_creatable_sockets | Number of simultaneous sockets supported. |
| uint32_t | curr_cmd_port | Current UART instance index for AT commands. |
| uint32_t | curr_data_port | Current UART instance index for data. |
| uint8_t | cmd_rx_queue_buf[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE] | Command port receive buffer used by byte queue // FreeRTOS. |
| StreamBufferHandle_t | socket_byteq_hdl | Socket stream buffer handle. |
| StaticStreamBuffer_t | socket_byteq_struct | Structure to hold stream buffer info. |
| volatile uint32_t | curr_socket_index | Currently active socket instance. |
| uint8_t | cmd_tx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_TX_BUF_SIZE] | Command send buffer. |
| uint8_t | cmd_rx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE] | Command receive buffer. |
| uint32_t | at_cmd_mode | Current command mode. |
| uint8_t | curr_ipaddr[4] | Current IP address of module. |
| uint8_t | curr_subnetmask[4] | Current Subnet Mask of module. |

| | | |
|---|---|---|
| uint8_t | curr_gateway[4] | Current GAteway of module. |
| SemaphoreHandle_t | tx_sem | Transmit binary semaphore handle. |
| SemaphoreHandle_t | rx_sem | Receive binary semaphore handle. |
| uint8_t | last_data[WIFI_ONCHIP_SILEX_RETURN_TEXT_LENGTH] | Tailing buffer used for command parser. |
| uart_instance_t * | uart_instance_objects[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | UART instance objects. |
| SemaphoreHandle_t | uart_tei_sem[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS] | UART transmission end binary semaphore. |
| ulpgn_socket_t | sockets[WIFI_ONCHIP_SILEX_CFG_NUM_CREATEABLE_SOCKETS] | Internal socket instances. |

## Enumeration Type Documentation

### ◆ sx_ulpgn_socket_status_t

| |
|---|
| enum sx_ulpgn_socket_status_t |
| Silex ULPGN Wifi socket status types |

### ◆ sx_ulpgn_socket_rw

| |
|---|
| enum sx_ulpgn_socket_rw |
| Silex socket shutdown channels |

### ◆ wifi_onchip_silex_sntp_enable_t

| |
|---|
| enum wifi_onchip_silex_sntp_enable_t |
| Silex WiFi module enable/disable for SNTP |

### ◆ wifi_onchip_silex_sntp_daylight_savings_enable_t

| |
|---|
| enum wifi_onchip_silex_sntp_daylight_savings_enable_t |
| Silex WiFi module enable/disable for SNTP |

## Function Documentation

### ◆ rm_wifi_onchip_silex_open()

fsp_err_t rm_wifi_onchip_silex_open ( wifi_onchip_silex_cfg_t const *const *p_cfg*)

Opens and configures the WIFI_ONCHIP_SILEX Middleware module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to pin configuration structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX successfully configured. |
| FSP_ERR_ASSERTION | The parameter p_cfg or p_instance_ctrl is NULL. |
| FSP_ERR_OUT_OF_MEMORY | There is no more heap memory available. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ rm_wifi_onchip_silex_close()

fsp_err_t rm_wifi_onchip_silex_close ( )

Disables WIFI_ONCHIP_SILEX.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX closed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ rm_wifi_onchip_silex_disconnect()

fsp_err_t rm_wifi_onchip_silex_disconnect ( )

Disconnects from connected AP.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | WIFI_ONCHIP_SILEX disconnected successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ rm_wifi_onchip_silex_socket_connected()

fsp_err_t rm_wifi_onchip_silex_socket_connected ( fsp_err_t * *p_status*)

Check if a specific socket instance is connected.

**Parameters**

| | | |
|---|---|---|
| [out] | p_status | Pointer to integer holding the socket connection status. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The parameter p_status is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |

### ◆ rm_wifi_onchip_silex_network_info_get()

fsp_err_t rm_wifi_onchip_silex_network_info_get ( uint32_t * *p_ip_addr*, uint32_t * *p_subnet_mask*, uint32_t * *p_gateway* )

Return the network information for the connection to the access point.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ip_addr | Pointer to integer holding the IP address. |
| [out] | p_subnet_mask | Pointer to integer holding the subnet mask. |
| [out] | p_gateway | Pointer to integer holding the gateway. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | A parameter pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_WIFI_AP_NOT_CONNECTED | No connection to access point has happened. |

### ◆ rm_wifi_onchip_silex_connect()

fsp_err_t rm_wifi_onchip_silex_connect ( const char * *p_ssid*, WIFISecurity_t *security*, const char * *p_passphrase* )

Connects to the specified Wifi Access Point.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ssid | Pointer to SSID of Wifi Access Point. |
| [in] | security | Security type to use for connection. |
| [in] | p_passphrase | Pointer to the passphrase to use for connection. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The parameter p_ssid or p_passphrase is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | No commas are accepted in the SSID or Passphrase. |

### ◆ rm_wifi_onchip_silex_mac_addr_get()

fsp_err_t rm_wifi_onchip_silex_mac_addr_get ( uint8_t * *p_macaddr*)

Get MAC address.

**Parameters**

| | | |
|---|---|---|
| [out] | p_macaddr | Pointer array to hold mac address. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The parameter p_macaddr is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_scan()

fsp_err_t rm_wifi_onchip_silex_scan ( WIFIScanResult_t * *p_results*, uint32_t *maxNetworks* )

Get the information about local Wifi Access Points.

**Parameters**

| | | |
|---|---|---|
| [out] | p_results | Pointer to a structure array holding scanned Access Points. |
| [in] | maxNetworks | Size of the structure array for holding APs. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The parameter p_results or p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_ping()

fsp_err_t rm_wifi_onchip_silex_ping ( uint8_t * *p_ip_addr*, uint32_t *count*, uint32_t *interval_ms* )

Ping an IP address on the network.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ip_addr | Pointer to IP address array. |
| [in] | count | Number of pings to attempt. |
| [in] | interval_ms | Interval between ping attempts. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The parameter p_ip_addr is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_ip_addr_get()

fsp_err_t rm_wifi_onchip_silex_ip_addr_get ( uint32_t * *p_ip_addr*)

Get the assigned module IP address.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ip_addr | Pointer an array to hold the IP address. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The parameter p_ip_addr is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_avail_socket_get()

fsp_err_t rm_wifi_onchip_silex_avail_socket_get ( uint32_t * *p_socket_id*)

Get the next available socket ID.

**Parameters**

| | | |
|---|---|---|
| [out] | p_socket_id | Pointer to an integer to hold the socket ID. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The parameter p_socket_id is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_WIFI_FAILED | Error occurred in the execution of this function |

### ◆ rm_wifi_onchip_silex_socket_status_get()

fsp_err_t rm_wifi_onchip_silex_socket_status_get ( uint32_t *socket_no*, uint32_t * *p_socket_status* )

Get the socket status.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [out] | p_socket_status | Pointer to an integer to hold the socket status |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl or p_socket_status is NULL. The value of socket_no is greater than/equal num_creatable_sockets. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_shutdown()

int32_t rm_wifi_onchip_silex_tcp_shutdown ( uint32_t *socket_no*, uint32_t *shutdown_channels* )

Shutdown portion of a socket

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | shutdown_channels | Specify if read or write channel is shutdown for socket |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl is NULL. The value of socket_no is greater than/equal num_creatable_sockets. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_socket_create()

fsp_err_t rm_wifi_onchip_silex_socket_create ( uint32_t *socket_no*, uint32_t *type*, uint32_t *ipversion* )

Create a new socket instance.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | type | Socket type. |
| [in] | ipversion | Socket IP type. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_tcp_connect()

fsp_err_t rm_wifi_onchip_silex_tcp_connect ( uint32_t *socket_no*, uint32_t *ipaddr*, uint32_t *port* )

Connect to a specific IP and Port using socket.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | ipaddr | IP address for socket connection. |
| [in] | port | Port number for socket connection. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_udp_connect()

fsp_err_t rm_wifi_onchip_silex_udp_connect ( uint32_t *socket_no*, uint32_t *ipaddr*, uint32_t *port*, uint32_t *type* )

Connect to a specific IP and Port using UDP socket.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | ipaddr | IP address for socket connection. 0 if type is server. |
| [in] | port | Port number for socket connection. |
| [in] | type | Type of UDP connection. Can be client or server. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_send()

int32_t rm_wifi_onchip_silex_send ( uint32_t *socket_no*, const uint8_t * *p_data*, uint32_t *length*, uint32_t *timeout_ms* )

Send data over TCP to a server.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [in] | p_data | Pointer to data to send. |
| [in] | length | Length of data to send. |
| [in] | timeout_ms | Timeout to wait for transmit end event |

**Return values**

| | |
|---|---|
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_instance_ctrl or parameter p_data is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |

### ◆ rm_wifi_onchip_silex_recv()

int32_t rm_wifi_onchip_silex_recv ( uint32_t *socket_no*, uint8_t * *p_data*, uint32_t *length*, uint32_t *timeout_ms* )

Receive data over TCP from a server.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID number. |
| [out] | p_data | Pointer to data received from socket. |
| [in] | length | Length of data array used for receive. |
| [in] | timeout_ms | Timeout to wait for data to be received from socket. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_ASSERTION | The p_instance_ctrl or parameter p_data is NULL. |

### ◆ rm_wifi_onchip_silex_socket_disconnect()

fsp_err_t rm_wifi_onchip_silex_socket_disconnect ( uint32_t *socket_no*)

Disconnect a specific socket connection.

**Parameters**

| | | |
|---|---|---|
| [in] | socket_no | Socket ID to disconnect |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | Bad parameter value was passed into function. |

### ◆ rm_wifi_onchip_silex_dns_query()

fsp_err_t rm_wifi_onchip_silex_dns_query ( const char * *p_textstring*, uint8_t * *p_ip_addr* )

Initiate a DNS lookup for a given URL.

**Parameters**

| | | |
|---|---|---|
| [in] | p_textstring | Pointer to array holding URL to query from DNS. |
| [out] | p_ip_addr | Pointer to IP address returned from look up. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function completed successfully. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_ASSERTION | The p_instance_ctrl, p_textstring, p_ip_addr is NULL. |
| FSP_ERR_NOT_OPEN | The instance has not been opened. |
| FSP_ERR_INVALID_ARGUMENT | The URL passed in is to long. |

### ◆ RM_WIFI_ONCHIP_SILEX_EpochTimeGet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_EpochTimeGet ( time_t * *p_utc_time*)

This will retrieve time info from an NTP server at the address entered via an during configuration. If the server isn't set or the client isn't enabled, then it will return an error. The date/time is retrieved as the number of seconds since 00:00:00 UTC January 1, 1970

**Parameters**

| | | |
|---|---|---|
| [out] | p_utc_time | Returns the epoch time in seconds. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully retrieved the system time from module. |
| FSP_ERR_ASSERTION | The parameter utc_time or p_instance_ctrl is NULL. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

Get the current system time as the number of seconds since epoch 1970-01-01 00:00:00 UTC

### ◆ RM_WIFI_ONCHIP_SILEX_LocalTimeGet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_LocalTimeGet ( uint8_t * *p_local_time*, uint32_t *size_string* )

Get the current local time based on current timezone in a string . Exp: Wed Oct 15 1975 07:06:00

**Parameters**

| | | |
|---|---|---|
| [out] | p_local_time | Returns local time in string format. |
| [in] | size_string | Size of p_local_time string buffer.The size of this string needs to be at least 25 bytes |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully returned the local time string. |
| FSP_ERR_ASSERTION | The parameter local_time or p_instance_ctrl is NULL. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | String size value passed in exceeds maximum. |

Get the current local time based on current timezone in a string format

### ◆ RM_WIFI_ONCHIP_SILEX_SntpEnableSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpEnableSet ( wifi_onchip_silex_sntp_enable_t *enable*)

Set the SNTP Client to Enable or Disable

**Parameters**

| | | |
|---|---|---|
| [in] | enable | Can be set to enable/disable for SNTP support. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

Enable or Disable the SNTP Client Service

### ◆ RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet ( uint8_t * *p_ip_address*)

Set the SNTP Client Server IP Address

**Parameters**

| | | |
|---|---|---|
| [in] | p_ip_address | Pointer to IP address of SNTP server in byte array format. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_ASSERTION | The parameter p_ip_address or p_instance_ctrl is NULL. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |

Update the SNTP Server IP Address

### ◆ RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet ( int32_t *hours*, uint32_t *minutes*, wifi_onchip_silex_sntp_daylight_savings_enable_t *daylightSavingsEnable* )

Set the SNTP Client Timezone

**Parameters**

| | | |
|---|---|---|
| [in] | hours | Number of hours (+/-) used for timezone offset from GMT. |
| [in] | minutes | Number of minutes used for timezone offset from GMT. |
| [in] | daylightSavingsEnable | Enable/Disable daylight saving in the timezone calculation. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully set the value. |
| FSP_ERR_WIFI_FAILED | Error occurred with command to Wifi module. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_ARGUMENT | Parameter passed into function was invalid. |

Update the SNTP Timezone

## 5.2.12.34 lwIP Baremetal Porting Layer (rm_lwip_sys_baremetal)

Modules » Networking

# Overview

This module provides system timer feature for baremetal.

Please refer to the lwIP documentation for further details.

### Features

This porting layer includes the following features.

- Timer
- Critical sections
- Compiler/Platform abstraction.

### 5.2.12.35 lwIP Ethernet Driver (rm_lwip_ether)

Modules » Networking

#### Functions

| | |
|---|---|
| err_t | rm_lwip_ether_init (struct netif *p_netif) |
| void | rm_lwip_ether_callback (ether_callback_args_t *p_args) |

#### Detailed Description

# Overview

This module provides a lwIP driver that is implemented using the Ethernet Interface.

Please refer to the lwIP documentation for further details.

#### Features

- Packet Types Supported
  - ARP
  - IPv4
  - IPv6
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT estimation fast recovery/fast retransmit and sending SACKs
- raw/native API for enhanced performance
- Link status callback

# Configuration

#### Configurations for Networking > lwIP Wrapper to (rm_lwip_ether)

| Configuration | Options | Default | Description |
|---|---|---|---|
| RTOS | | | |
| RX thread stacksize | Must be a positive integer | 1024 | Stack size of RX thread. |
| RX thread priority | Must be a positive integer | 4 | Priority of RX thread. |
| Name | Name must be a valid C symbol | g_lwip_ether0 | Module name. |
| MTU | Must be an integer from 576 to 1500 | 1500 | MTU size. |
| Netif flags | • UP<br>• BROADCAST | • BROADCAST<br>• ETHARP | Flags representing the capabilities of the |

| | | | |
|---|---|---|---|
| | • LINK_UP<br>• ETHARP<br>• ETHERNET<br>• IGMP<br>• MLD6 | | Ethernet device |
| Zero-copy mode | • Disable<br>• Enable | Disable | Enable or disable zero-copy mode. |
| RX buffer pool size | Must be a positive integer | 10 | Buffer pool size for zero-copy RX. |
| Link check interval | Must be a positive integer | 100 | Interval for checking link status. |

# Usage Notes

### Limitations

# Examples

### Basic Example

This is a basic example of minimal use of the lwIP Ether Driver in an application.

```
#include "lwip/tcp.h"
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#define LWIP_EXAMPLE_PORT_NUMBER 9000
/* TCP callback functions. */
static err_t lwip_example_connected_callback(void * arg, struct tcp_pcb * tpcb, err_t
err);
err_t       lwip_example_sent_callback(void * arg, struct tcp_pcb * tpcb, u16_t len);
err_t       lwip_example_recv_callback(void * arg, struct tcp_pcb * tpcb, struct pbuf
* p, err_t err);
volatile uint8_t g_lwip_example_connected_flag;
volatile uint8_t g_lwip_example_sent_flag;
volatile uint8_t g_lwip_example_recv_flag;
void rm_lwip_ether_example () {
 struct netif    netif;
 struct tcp_pcb * pcb = NULL;
    ip_addr_t ipaddr;
```

```
    ip_addr_t netmask;
    ip_addr_t gw;
    ip_addr_t server_ip;
 const char data[] = "lwIP Hello world.";
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
 /* Create a pcb for client. */
    pcb = tcp_new();
 /* Start connection to the server. */
    g_lwip_example_connected_flag = false;
    IP4_ADDR(&server_ip, 192, 168, 10, 1);
    tcp_connect(pcb, &server_ip, LWIP_EXAMPLE_PORT_NUMBER,
lwip_example_connected_callback);
 /* Waiting for connection to be established. */
 while (false == g_lwip_example_connected_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Set callback for receiving complete. */
    g_lwip_example_recv_flag = false;
    tcp_recv(pcb, lwip_example_recv_callback);
```

```
 /* Set callback for sending complete. */
    g_lwip_example_sent_flag = false;
    tcp_sent(pcb, lwip_example_sent_callback);
 /* Send a message. */
    tcp_write(pcb, data, sizeof(data), TCP_WRITE_FLAG_COPY);
 while (false == g_lwip_example_sent_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 while (false == g_lwip_example_recv_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Close TCP client and netif. */
    tcp_close(pcb);
    netif_remove(&netif);
}
err_t lwip_example_connected_callback (void * arg, struct tcp_pcb * tpcb, err_t err)
{
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(err);
 FSP_PARAMETER_NOT_USED(tpcb);
    g_lwip_example_connected_flag = true;
 return err;
}
err_t lwip_example_sent_callback (void * arg, struct tcp_pcb * tpcb, u16_t len) {
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(tpcb);
 FSP_PARAMETER_NOT_USED(len);
    g_lwip_example_sent_flag = true;
 return ERR_OK;
}
```

```
err_t lwip_example_recv_callback (void * arg, struct tcp_pcb * tpcb, struct pbuf * p,
err_t err) {
 FSP_PARAMETER_NOT_USED(arg);
 FSP_PARAMETER_NOT_USED(tpcb);
 FSP_PARAMETER_NOT_USED(err);
    g_lwip_example_recv_flag = true;
 /* If the callback function returns ERR_OK it must have freed the pbuf, otherwise it
must not have freed it. */
 if (p != NULL)
    {
       pbuf_free(p);
    }
 return ERR_OK;
}
```

### Function Documentation

#### ◆ rm_lwip_ether_init()

| err_t rm_lwip_ether_init ( struct netif * *p_netif*) |
|---|
| Initialize ethernet hardware and lwIP network interface. This function is passed to netif_add() |

#### ◆ rm_lwip_ether_callback()

| void rm_lwip_ether_callback ( ether_callback_args_t * *p_args*) |
|---|
| Callback of Ethernet interrupt subroutine. This function is set to ethernet driver callback by configurator. |

### 5.2.12.36 lwIP FreeRTOS Porting Layer (rm_lwip_sys_freertos)

Modules » Networking

# Overview

This module provides a lwIP porting layer for FreeRTOS.

Please refer to the lwIP documentation for further details.

#### Features

This porting layer includes the following features.

- Compiler/Platform abstraction.

All other features of FreeRTOS porting layer are provided by the lwIP library.

### 5.2.12.37 lwIP HTTP Server

Modules » Networking

This module provides the lwIP HTTP Server integration documentation.

## Overview

The LwIP HTTP Server library can be used to respond HTTP request. The documentation for the library can be found at the following link: lwIPServer.

#### Features

The LwIP HTTP Server:

- Can respond from HTTP GET and POST requests.
- Supports Server-Side Includes (SSI) and Common Gateway Interface (CGI).
- Can set up TLS connection handling (HTTPS).

## Configuration

#### Build Time Configurations for Source

The following build time configurations are defined in lwip/lwip/apps/httpd_opts.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| LwIP HTTPD CGI (old style) | • Enabled<br>• Disabled | Disabled | Enable CGI support (old style). Use this CGI type e.g. to execute specific actions and return a page that does not depend on the CGI parameters. |
| LwIP HTTPD CGI SSI (new style) | • Enabled<br>• Disabled | Disabled | Enable CGI support with SSI feature (new style). Use this CGI handler if you want CGI information passed on to SSI. |

| | | | |
|---|---|---|---|
| LwIP HTTPD SSI | • Enabled<br>• Disabled | Disabled | Enable SSI support (Server-Side-Includes). |
| LwIP HTTPD SSI Raw | • Enabled<br>• Disabled | Disabled | Enable to implement an SSI tag handler callback that gets a const char* to the tag (instead of an index into a pre-registered array of known tags). |
| LwIP HTTPD SSI By File Extension | • Overridden<br>• Enabled<br>• Disabled | Enabled | enable/disable or override the file SSI tag scan function. You need to provide an external function to perform the check: u8_t http_uri_is_ssi(struct fs_file *file, const char *uri) |
| LwIP HTTPD SSI Extension | Manual Entry | ".shtml", ".shtm", ".ssi", ".xml", ".json" | List of file extensions handled as SSI files. Only used if LWIP_HTTPD_SSI_BY_FILE_EXTENSION != 0. |
| LwIP HTTPD Support POST | • Enabled<br>• Disabled | Disabled | Set this to 1 to support HTTP POST. |
| LwIP HTTPD Maximum CGI Parameters | Must be a non zero positive interger. | 16 | The maximum number of parameters that the CGI handler can be sent. |
| LwIP HTTPD SSI Multipart | • Enabled<br>• Disabled | Disabled | SSI handler function is called with 2 more arguments to handle string too long to be inserted at once |
| LwIP HTTPD Maximum Tag Name Length | Must be a non zero positive interger. | 8 | The maximum length of the string comprising the SSI tag name. The longer tags are ignored. |
| LwIP HTTPD Maximum Tag Insert Length | Must be a non zero positive interger. | 192 | The maximum length of string that can be returned to replace any given tag. |
| LwIP HTTPD POST Manual WND | • Enabled<br>• Disabled | Disabled | Enable configurating POST window response for HTTP server |
| HTTPD Server Agent | Manual Entry | lwIP/" LWIP_VERSION_STRING | This string is passed in the HTTP header as |

| | | | |
|---|---|---|---|
| | | "<br>(<br>http://savannah.nongnu.org/projects/lwip) | 'Server: HTTPD_SERVER_AGENT'. |
| LwIP HTTPD Dynamic Headers | • Enabled<br>• Disabled | Disabled | Set this to 1 if you want to include code that creates HTTP headers at runtime. Else use static header created by the makefsdata tool. |
| HTTPD Debug | • Disabled<br>• Enabled | Disabled | Configure debug for HTTPD file send timing |
| HTTPD Use Mem Pool | • Enabled<br>• Disabled | Disabled | Set this to 1 to allocate http_state instead of using heap.If enabled, define MEMP_NUM_PARALLEL_HTTPD_CONNS (and MEMP_NUM_PARALLEL_HTTPD_SSI_CONNS for SSI) to set the size of the pool(s). |
| HTTPD Server Port | Must be an interger in range 0-65535. | 80 | The http server port for HTTPD to use |
| HTTPD Server Port HTTPS | Must be an interger in range 0-65535. | 443 | The https server port for HTTPD to use |
| HTTPD Enable HTTPS | • Enabled<br>• Disabled | Disabled | Enable https support. |
| HTTPD Maximum Retries | Must be a positive integer. | 4 | Maximum retries before the connection is aborted/closed. |
| HTTPD Poll Interval | Must be a non zero positive interger. | 4 | How long the Server will keep the connection. The poll delay is X*500ms. |
| HTTPD TCP Priority | Must be an interger in range 1-127. | 1 | Priority for tcp pcbs created by HTTPD 1 by default range from 1-127. |
| LwIP HTTPD Timing | • Enabled<br>• Disabled | Disabled | Set this to 1 to enable timing each file sent. |
| HTTPD Debug Timing | • Disabled<br>• Enabled | Disabled | Set this to 1 to enable timing each file sent. |
| LwIP HTTPD Support Extstatus | • Enabled<br>• Disabled | Disabled | Set this to one to show error pages when parsing request fail. Else simply close the connection |

| | | | |
|---|---|---|---|
| LwIP HTTPD Support V09 | • Enabled<br>• Disabled | Enabled | Set this to 0 to drop support for HTTP/0.9 clients (to save some bytes). |
| LwIP HTTPD Support 11 Keepalive | • Enabled<br>• Disabled | Disabled | Set this to 1 to enable HTTP/1.1 persistent connections. |
| LwIP HTTPD Support Requestlist | • Enabled<br>• Disabled | Enabled | Set this to 1 to support HTTP request coming in in multiple packets/pbufs. |
| LwIP HTTPD Request Queuelen | Must be a non zero positive interger. | 5 | Number of rx pbufs to enqueue to parse an incoming request. |
| LwIP HTTPD Request Bufsize | Must be a non zero positive interger. | LWIP_HTTPD_MAX_REQ_LENGTH | Number of (TCP payload-) bytes (in pbufs) to enqueue to parse and incoming request. |
| LwIP HTTPD Maximum Request Length | Must be a non zero positive interger. | LWIP_MIN(1023, (LWIP_HTTPD_REQ_QUEUELEN * PBUF_POOL_BUFSIZE)) | Defines the maximum length of a HTTP request line. |
| LwIP HTTPD Maximum Request URI Length | Must be a positive interger. | 63 | This is the size of a static buffer used when URIs end with '/'. |
| LwIP HTTPD POST Maximum Response URI Length | Must be a non zero positive interger. | 63 | Maximum length of the filename to send as response to a POST request. |
| LwIP HTTPD SSI Include Tag | • Enabled<br>• Disabled | Enabled | Set this to 0 to not send the SSI tag. |
| LwIP HTTPD Abort On Close Memory Error | • Enabled<br>• Disabled | Disabled | Set this to 1 to call tcp_abort when tcp_close fails with memory error. |
| LwIP HTTPD Kill Old On Connections Exceeded | • Enabled<br>• Disabled | Disabled | Set this to 1 to kill the oldest connection. |
| LwIP HTTPD Omit Header For Extensionless URI | • Enabled<br>• Disabled | Disabled | Set this to 1 to send URIs without extension without headers. |
| HTTPD Limit Sending To 2mss | • Enabled<br>• Disabled | Enabled | By default, the httpd is limited to send 2*pcb->mss. |
| HTTPD Maximum Write Length | The name function must be a valid C symbol | altcp_mss | Define this to a function that returns the maximum amount |

| | | | |
|---|---|---|---|
| | | | of data to enqueue. The function have this signature: u16_t fn(struct altcp_pcb* pcb);. |
| LwIP HTTPD Custom Files | • Enabled<br>• Disabled | Disabled | Set this to 1 and provide the functions: - int fs_open_custom(struct fs_file *file, const char *name) Called first for every opened file to allow opening files that are not included in fsdata(_custom).c - void fs_close_custom(struct fs_file *file) Called to free resources allocated by fs_open_custom(). |
| LwIP HTTPD Dynamic File Read | • Enabled<br>• Disabled | Disabled | Set this to 1 to support fs_read(). Else only one-block files are supported, and the contents must be ready after fs_open(). |
| LwIP HTTPD File State | • Enabled<br>• Disabled | Disabled | Set this to 1 to include an application stateargument per file that is opened. |
| LwIP HTTPD File Extension | • Enabled<br>• Disabled | Disabled | Set this to 1 to add the pextension field to the fs_file structure.New code should use LWIP_ HTTPD_FILE_STATE instead. |
| HTTPD Precalculated Checksum | • Enabled<br>• Disabled | Disabled | Support asynchronous read operations. |
| LwIP HTTPD FS Async Read | • Enabled<br>• Disabled | Disabled | Set this to 1 to add the pextension field. |
| HTTPD Fsdata File | Manual Entry | fsdata_sample.c | Provide source name file of FS custom data. This data file will be included by fs.c file |
| HTTP Is Data Volatile | • Allow Copy and More<br>• Allow Copy<br>• Undefined | Allow Copy | Determines if HTTP data should be copied into the TCP buffer before transmission. |

| | | | |
|---|---|---|---|
| | | | Allow Copy: data will be copied into memory belonging to the stack.<br>Allow Copy and More: Notify to TCP that data will be copied into memory belonging to the stack<br>and additional data will follow in subsequent TCP transmissions. |
| LwIP HTTPD Use Default Template | • Enabled FSdata Generator<br>• Disabled FSdata Generator | Enabled FSdata Generator | Set this to 1 to when intent to use the default template from the generator |

# Usage Notes

### Limitations

- In normal mode, the user must select the "HTTP Is Data Volatile" option as Allow Copy mode (TCP_WRITE_FLAG_COPY) in the FSP Configurator properties to ensure tcp_write() functions correctly.
- When using the Server-Side Includes (SSI) mode or the Common Gateway Interface (CGI) mode, make sure that the "HTTP Is Data Volatile" option is set to Allow Copy and More mode (TCP_WRITE_FLAG_COPY | TCP_WRITE_FLAG_MORE) in the FSP Configurator properties to work properly.

### How to use fsdata file

There are two ways to use fsdata file as follows:

### 1. Use the available fsdata_sample.c file

When the stack "LwIP HTTP Server" is added, by default it includes the stack "LwIP HTTP Server FS" which is the sample file provided in the LwIP HTTP Server library repository, and the fsdata_sample.c file is generated at the src/fsdata/ path.

### 2. Generate new fsdata.c file using makefsdata script

In case, creating a new fsdata.c file by makefsdata script, follow these detailed steps:

*Note*

*Reffer to this link readme.txt for an introduction by the author about the makefsdata scripts.*

- ***Setup Environment***
  1. *Make sure Perl is installed on your development environment to ensure the required script runs.*
- ***Convert Server Contents to C Source Code***
  1. *Create a new folder. (the created folder can be placed outside the project workspace).*
  2. *Copy the "makefsdata" folder from the LwIP HTTP Server library repository into the created folder.*
  3. *Inside the created folder, create a new subfolder named "fs" (the name "fs" is mandatory).*
  4. *Place the files to be converted into the "fs" folder.*
  5. *Open a terminal on Linux or Git Bash/Windows Command Prompt on Windows. (if Cygwin*

*is installed) and navigate to the created folder.*
6. *Type the command "perl makefsdata".*
7. *The new "fsdata.c" file will be generated in the created folder.*

- ***Configure the Generated File***
  1. *Remove the the stack "LwIP HTTP Server FS".*
  2. *Add an "include guard" to prevent multiple inclusions in the new file "fsdata.c".*
  3. *Include the headers "lwip/apps/fs.h" and "lwip/def.h" in the new file "fsdata.c".*
  4. *Copy only the new file "fsdata.c" into the project workspace.*
  5. *In the FSP Configurator properties, set the "HTTPD Fsdata File" option to the new file "fsdata.c".*

### TLS connection

The stack and heap sizes must be properly configured when using the TLS layer with MbedTLS. Depending on the specific configuration and use case, these values may need to be increased from their defaults. For more information on memory requirements, refer to the **Memory Usage** section in Mbed Crypto H/W Acceleration (rm_psa_crypto).

*Note*

*When using hardware acceleration for the MbedTLS implementation, verify that the target device supports all required mbedCrypto features for the TLS layer of lwIP. Refer to the **HW Overview** and **Features** sections in Mbed Crypto H/W Acceleration (rm_psa_crypto) to check hardware support. If any required feature is not supported, configure the corresponding features to use the software crypto implementation instead.*

# Examples

### Basic Example

This is a basic example of minimal use of the LwIP HTTP Server APIs in an application.

```
#if HTTPD_ENABLE_HTTPS
 #include "lwip/altcp_tls.h"
 #include "mbedtls/platform.h"
#endif
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#include "lwip/apps/httpd.h"
#if HTTPD_ENABLE_HTTPS
/* Example server key, needs to be updated. */
static uint8_t gs_server_key[] =
 "-----BEGIN PRIVATE KEY-----\n"
 "MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDKrJdfXTLDsPb4\n"
 "7ACEg7kfMYuISDOaIgTlZBUSfAbeozju5zZoTigOTbJG4oUgV52qkoumSQ8L6c80\n"
 "RA+0hnslzuYXo2mz3zdlLBL1PZwXOvQIJ1IdveYIiH7DIsEiQ5o1wvPz7RrKCUGY\n"
```

```
 "CfhDLFfxgZUDQ51p31V1KgU92pIgEJUfrmKyYT1HK3dQC9QP4AkmUrADmZInhzku\n"
 "NktnE9/aZAC1DqVH+bcIizT4WJDY7wY1lRAM9GghRhJOOGPDbBfCrUsTd7qOd1pT\n"
 "BPGTnGVskrmC2YsO+Xtc0Mbh+26V6rfES7QmKtlLPDnApnyEPe4EzlluMhXLKI6M\n"
 "UhGWRp8TAgMBAAECggEBAKdS2h33OaPqA9v6K2s7vKgOJV48Da4gi94+RE9gaeOA\n"
 "6c4CwR5X+CXm61ouRS4wZ5CdMAH2qx9wxool7rlhlWXfbILzfJ7tmDREsNobJTTg\n"
 "jPv2SosXbkR5O/4qQTzihvv/mZw2kjlebvLwuC/s+iX+j2GEfv8bTNhP+sfYHW+u\n"
 "CL11deCRosu5GHe5Jgs2YSGmi1sMrrzAMBIq/K6SkjzM+Ty9e6AhW0/EAkqeHZmU\n"
 "jUT7MVgfATpf1jBe5Blgu0/ZRixdb01PeOxxEmqOdjl6sKLneczNS7i1yUsAjR7J\n"
 "dlLjNvp7bz91zl08BoRBk/XdkNKkWuVA5Eo/XMotcLECgYEA64i9rymLb5EVbP56\n"
 "a9MJVnmQQ3IA3nX26Vi3ksHyzSeegazA3xQxHM2R99dNJNvXP4OkNyL1TZQVVhsb\n"
 "rHR9QFMk1WxoLmw42krba1gdQ8UMh/C21HN7Q7eyVDYoEWHOBoKWQoZI6SFgLx8W\n"
 "S4CRZObExsRxtknHr7NUuFiINrUCgYEA3EjofZrzENzzZUzzsKVm2q8R6GIhYC06\n"
 "5VFcBEsoz1NNzyf93iLDwCluvUqt8jsaQsjNDpwdeSe273f80Hp3xqe7ADg5O6rK\n"
 "S2ZMmWpcuugMI/l9KnhLmQmFz62phL4ioT5ojIUBhs8J/oXrldv1K/7ZkuOj0179\n"
 "v6vhsW2dk6cCgYABskJreIkgjJLy7gJDtIlkrgtz8OZAs+jLVXA9BliRUa+6j1v0\n"
 "P1j7NB/RASP1GAA83N3Z586MeHyO2jMg0tX0GKUzREybyPu/1rot07yiOFEWyY0H\n"
 "iq1gl3oZUSqItYOeWGJsqDruMudWb4rhkFN7vo28QL3w6iceBNSfX0Pb4QKBgEs4\n"
 "jkNN5RBuEqMV2uUIJg4o2BmdYL5B06m41Y4VyDK7xIcnpdXz0xpMS6dR2AIuUnJZ\n"
 "tUP25U2Y1SwCsg7SsxHf9shFaq3ITVnf7eo5Wr7sPDEQocAd4dq5WLspFwjgEF01\n"
 "PWg2huXszwHd4yuCN781QECbgTbNaiLIY8I9pyBjAoGBAIY8y7uN+pU2LdW7LG56\n"
 "E6wxhSl9Epz1PV31W93gQyOfY8RZSZ3upbe4uylg2ONe3m7cLB3cPg4ohSxM84RI\n"
 "60jvXHWPi+DG0Ub0Y3E1O6ZAW1UnrE3Q+J01bhNK8oy3X4sht9dg6VFZK8UFCk4V\n"
 "cPZoDs+ykQpvVD8dnII31qAw\n"
 "-----END PRIVATE KEY-----";
/* Example server certificate, needs to be updated. */
static uint8_t g_server_crt[] =
 "-----BEGIN CERTIFICATE-----\n"
 "MIIDbTCCAlWgAwIBAgIUKvNkxPfd58IpndbgRLZetCiVS84wDQYJKoZIhvcNAQEL\n"
 "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
 "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAgFw0yMDA5MTExNjE5NThaGA8yMjAw\n"
 "MDIxNjE2MTk1OFowRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUx\n"
 "ITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDCCASIwDQYJKoZIhvcN\n"
 "AQEBBQADggEPADCCAQoCggEBAMqsl19dMsOw9vjsAISDuR8xi4hIM5oiBOVkFRJ8\n"
 "Bt6jOO7nNmhOKA5NskbihSBXnaqSi6ZJDwvpzzRED7SGeyXO5hejabPfN2UsEvU9\n"
```

```
 "nBc69AgnUh295giIfsMiwSJDmjXC8/PtGsoJQZgJ+EMsV/GBlQNDnWnfVXUqBT3a\n"
 "kiAQlR+uYrJhPUcrd1AL1A/gCSZSsAOZkieHOS42S2cT39pkALUOpUf5twiLNPhY\n"
 "kNjvBjWVEAz0aCFGEk44Y8NsF8KtSxN3uo53WlME8ZOcZWySuYLZiw75e1zQxuH7\n"
 "bpXqt8RLtCYq2Us8OcCmfIQ97gTOWW4yFcsojoxSEZZGnxMCAwEAAaNTMFEwHQYD\n"
 "VR0OBBYEFCL0XSgJ4bT7qqLtjdNHHfDhXoV5MB8GA1UdIwQYMBaAFCL0XSgJ4bT7\n"
 "qqLtjdNHHfDhXoV5MA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEB\n"
 "AIREt6x3cwQeRHaNkBjuYbOmB1LXa1az32d4nrvHG9aVfAOlIZuI2Zz3l3JJFT1M\n"
 "7NsSyhYojOgagEFs+pfgW6o6YG62e8Stne2J1wppGNqckLTJyeYnAFzKakfc9KXU\n"
 "Cq4v+JesOwzXmz3hbZywZ5rDJJHt/BoixL6C82fHUFNgzyH4dFpGWbndH29p8wr4\n"
 "hiTR4JxH5qlogMmWqnIJ26pOu26+xfPVhLy4o84MZl0e5QqRg05LBrFd5c8/BKxq\n"
 "Cr1qcy3Gc67/skwXMO1Dm+OSmDyGgvpAC0XXX1XpY3+/Y2yIT2glx09JsnyoXfES\n"
 "pasiCAXmwq48oPhw9PKoYCQ=\n"
 "-----END CERTIFICATE-----";
#endif
/* LwIP HTTPD example */
void lwip_httpd_example () {
 struct netif netif;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable. */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
```

```
#if !HTTPD_ENABLE_HTTPS
 /* Initialize the HTTP server. */
    httpd_init();
#else
 /* Setup the platform; initialize the SCE. */
 mbedtls_platform_setup(NULL);
 /* Create an TLS server configuratio.n */
 struct altcp_tls_config * configuration =
altcp_tls_create_config_server_privkey_cert((const u8_t *) gs_server_key,
 sizeof(gs_server_key),

     NULL,

     0,

     (const u8_t *) g_server_crt,
 sizeof(g_server_crt));
 /* Initialize the HTTPS server. */
    httpd_inits(configuration);
#endif
 /* Program main loop. */
 while (true)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
}
```

### 5.2.12.38 lwIP MQTT

Modules » Networking

This module provides the lwIP MQTT integration documentation.

# Overview

The lwIP MQTT library can connect to a third party MQTT broker such as Mosquitto. The documentation for the library can be found at the following link: lwIPMQTT.

### Features

- MQTT connections to MQTT broker
- Support set up TLS connection handling with one-way server authentication or two-way server/client authentication
- The version of MQTT protocol supported in lwIP is 3.1.1. Details of the MQTT protocol can be found at: mqtt-v3.1.1

# Usage Notes

### TLS connection

The stack and heap sizes must be properly configured when using the TLS layer with MbedTLS. Depending on the specific configuration and use case, these values may need to be increased from their defaults. For more information on memory requirements, refer to the **Memory Usage** section in Mbed Crypto H/W Acceleration (rm_psa_crypto).

*Note*

*When using hardware acceleration for the MbedTLS implementation, verify that the target device supports all required mbedCrypto features for the TLS layer of lwIP. Refer to the **HW Overview** and **Features** sections in Mbed Crypto H/W Acceleration (rm_psa_crypto) to check hardware support. If any required feature is not supported, configure the corresponding features to use the software crypto implementation instead.*

# Examples

### Basic Example

This is a basic example of minimal use of the lwIP MQTT APIs in an application.

```
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 #include "lwip/altcp_tls.h"
 #include "mbedtls/platform.h"
#endif
#include "lwip/tcp.h"
#include "lwip/timeouts.h"
#include "lwip/init.h"
#include "lwip/ip4.h"
#include "lwip/apps/mqtt.h"
#define LWIP_MQTT_EXAMPLE_PORT_NUMBER 8883
#define LWIP_MQTT_EXAMPLE_TOPIC "/lwip/example"
#define LWIP_MQTT_EXAMPLE_TOPIC_LENGTH 13
```

```
#define LWIP_MQTT_EXAMPLE_BUF_SIZE 50
#define LWIP_MQTT_EXAMPLE_KEEP_ALIVE 100
/* LwIP MQTT callback functions. */
static void lwip_mqtt_example_connection_callback(mqtt_client_t * client, void * arg,
mqtt_connection_status_t status);
static void lwip_mqtt_example_pub_sub_request_callback(void * arg, err_t result);
static void lwip_mqtt_example_incoming_publish_callback(void * arg, const char *
topic, u32_t tot_len);
static void lwip_mqtt_example_incoming_data_callback(void * arg, const u8_t * data,
u16_t len, u8_t flags);
volatile uint8_t g_lwip_mqtt_example_connected_flag;
volatile uint8_t g_lwip_mqtt_example_receive_incomming_msg_flag;
volatile uint8_t g_lwip_mqtt_example_pub_sub_flag;
static unsigned char g_mqtt_lwip_example_incoming_msg[LWIP_MQTT_EXAMPLE_BUF_SIZE];
static char g_mqtt_lwip_example_incoming_topic[LWIP_MQTT_EXAMPLE_BUF_SIZE];
/* LwIP MQTT connection information */
struct mqtt_connect_client_info_t lwip_mqtt_example_client_info =
{
 "lwip_mqtt_example",
    NULL,                               /* user */
    NULL,                               /* pass */
    LWIP_MQTT_EXAMPLE_KEEP_ALIVE,       /* keep alive */
    NULL,                               /* will_topic */
    NULL,                               /* will_msg */
    0,                                  /* will_qos */
    0                                   /* will_retain */
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    , NULL
#endif
};
#if LWIP_ALTCP && LWIP_ALTCP_TLS
/* Example ca certificate, needs to be updated. */
static uint8_t gs_ca_crt[] =
 "-----BEGIN CERTIFICATE-----\n"                                    \
```

```
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                   \
 "-----END CERTIFICATE-----";
#endif
void lwip_mqtt_example () {
 struct netif    netif;
    mqtt_client_t * g_lwip_mqtt_example_client;
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;
    ip_addr_t server_ip_address;
 const char data[] = "lwIP MQTT Hello world.";
 /* Initializing lwIP core. */
    lwip_init();
 /* Set ip address of the board. */
    IP4_ADDR(&ipaddr, 192, 168, 10, 4);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&gw, 192, 168, 10, 1);
```

```
 /* Initialize netif. Please pass rm_lwip_ether_init() function and the instance of
rm_lwip_ether to netif_add(). */
    netif_add(&netif, &ipaddr, &netmask, &gw, &g_lwip_ether0_instance,
rm_lwip_ether_init, netif_input);
 /* Set netif as default and enable */
    netif_set_default(&netif);
    netif_set_up(&netif);
    netif_set_link_up(&netif);
    IP4_ADDR(&server_ip_address, 192, 168, 10, 1);
 /* Initialize lwIP MQTT Client */
    g_lwip_mqtt_example_client = mqtt_client_new();
 /* Establish MQTT connection to server */
    g_lwip_mqtt_example_connected_flag = false;
#if LWIP_ALTCP && LWIP_ALTCP_TLS
 /* Create a TLS client configuration */
 struct altcp_tls_config * configuration =
       altcp_tls_create_config_client((const u8_t *) gs_ca_crt, sizeof(gs_ca_crt));
 /* Initializing TLS configuration */
    lwip_mqtt_example_client_info.tls_config = configuration;
#endif
    assert(ERR_OK ==
          mqtt_client_connect(g_lwip_mqtt_example_client, &server_ip_address,
LWIP_MQTT_EXAMPLE_PORT_NUMBER,
                              lwip_mqtt_example_connection_callback, NULL,
&lwip_mqtt_example_client_info));
 /* Waiting for connection to be established. */
 while (false == g_lwip_mqtt_example_connected_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Set callback for receiving incomming messsage from subscribed MQTT topic */
    g_lwip_mqtt_example_receive_incomming_msg_flag = false;
    mqtt_set_inpub_callback(g_lwip_mqtt_example_client,
```

```
                            lwip_mqtt_example_incoming_publish_callback,
                            lwip_mqtt_example_incoming_data_callback,
                            NULL);
 /* Subcribe to MQTT topic with Qos 0 */
    g_lwip_mqtt_example_pub_sub_flag = false;
    assert(ERR_OK ==
           mqtt_subscribe(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC, 0,
                          lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Waiting for subcribe process completed. */
 while (false == g_lwip_mqtt_example_pub_sub_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Publish message to MQTT topic with Qos 0 and retain flag disabled */
    g_lwip_mqtt_example_pub_sub_flag = false;
    assert(ERR_OK ==
           mqtt_publish(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC, data,
strlen(data), 0, 0,
                        lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Waiting for publish process completed. */
 while (false == g_lwip_mqtt_example_pub_sub_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Waiting for receive incomming process completed. */
 while (false == g_lwip_mqtt_example_receive_incomming_msg_flag)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       sys_check_timeouts();
    }
 /* Make sure receiving correct topic and message */
    assert(0 == memcmp(g_mqtt_lwip_example_incoming_topic, LWIP_MQTT_EXAMPLE_TOPIC,
```

```
sizeof(LWIP_MQTT_EXAMPLE_TOPIC)));
    assert(0 == memcmp(g_mqtt_lwip_example_incoming_msg, data, sizeof(data)));
 /* Unsubcribe to MQTT topic */
    assert(ERR_OK ==
          mqtt_unsubscribe(g_lwip_mqtt_example_client, LWIP_MQTT_EXAMPLE_TOPIC,
                           lwip_mqtt_example_pub_sub_request_callback, NULL));
 /* Close MQTT connection and netif. */
    mqtt_disconnect(g_lwip_mqtt_example_client);
    netif_remove(&netif);
    mqtt_client_free(g_lwip_mqtt_example_client);
}
/* LwIP MQTT callback functions */
void lwip_mqtt_example_incoming_publish_callback (void * arg, const char * topic,
u32_t tot_len)
{
 FSP_PARAMETER_NOT_USED(arg);
    memcpy(g_mqtt_lwip_example_incoming_topic, topic, tot_len);
}
void lwip_mqtt_example_incoming_data_callback (void * arg, const u8_t * data, u16_t
len, u8_t flags)
{
 FSP_PARAMETER_NOT_USED(arg);
 if (flags & MQTT_DATA_FLAG_LAST)
    {
       memcpy(g_mqtt_lwip_example_incoming_msg, data, len);
       g_lwip_mqtt_example_receive_incomming_msg_flag = true;
    }
}
void lwip_mqtt_example_connection_callback (mqtt_client_t * client, void * arg,
mqtt_connection_status_t status)
{
 FSP_PARAMETER_NOT_USED(client);
 FSP_PARAMETER_NOT_USED(arg);
 if (MQTT_CONNECT_ACCEPTED == status)
```

```
    {
       g_lwip_mqtt_example_connected_flag = true;
    }
}
void lwip_mqtt_example_pub_sub_request_callback (void * arg, err_t result)
{
 FSP_PARAMETER_NOT_USED(arg);
 if (ERR_OK == result)
    {
       g_lwip_mqtt_example_pub_sub_flag = true;
    }
}
```

## 5.2.13 Power

Modules

### Detailed Description

Power Modules.

### Modules

| | |
|---|---|
| | Low Power Modes (r_lpm)<br>Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface. |

### 5.2.13.1 Low Power Modes (r_lpm)

Modules » Power

### Functions

| | |
|---|---|
| fsp_err_t | R_LPM_Open (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| fsp_err_t | R_LPM_LowPowerReconfigure (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg) |
| | |

| | |
|---|---|
| fsp_err_t | R_LPM_LowPowerModeEnter (lpm_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LPM_IoKeepClear (lpm_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_LPM_Close (lpm_ctrl_t *const p_api_ctrl) |

### Detailed Description

Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface.

# Overview

The low power modes driver is used to configure and place the device into the desired low power mode. Various sources can be configured to wake from standby, request snooze mode, end snooze mode or end deep standby mode.

### Features

The LPM HAL module has the following key features:

- Supports the following low power modes:
  - Deep Software Standby mode (On supported MCUs)
  - Deep Sleep (On supported MCUs)
  - Software Standby mode
  - Sleep mode
  - Snooze mode (On supported MCUs)
- Supports reducing power consumption when in deep software standby mode through internal power supply control and by resetting the states of I/O ports.
- Supports disabling and enabling the MCU's other hardware peripherals

# Configuration

### Build Time Configurations for r_lpm

The following build time configurations are defined in fsp_cfg/r_lpm_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Standby Limit | • Enabled<br>• Disabled | Disabled | If enabled, standby configuration only applies in R_LPM_LowPowerModeEnter. Otherwise, standby configuration applies to any WFI call. |

### Configurations for Power > Low Power Modes (r_lpm)

This module can be added to the Stacks tab via New Stack > Power > Low Power Modes (r_lpm). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_lpm0 | Module name. |
| Low Power Mode | MCU Specific Options | | Power mode to be entered. |
| Output port state in standby and deep standby | MCU Specific Options | | Select the state of output pins during standby. Applies to address output, data output, and other bus control output pins. |
| Supply of SOSC clock to peripheral function in standby | MCU Specific Options | | If enabled, SOSC clock is provided to peripheral functions in Software Standby and Snooze modes. |
| Startup speed of the HOCO in Standby and Snooze modes | MCU Specific Options | | If enabled, the HOCO enters high-speed startup mode, shortening standby release time, and snooze transition time. |
| Flash mode in sleep or snooze | MCU Specific Options | | Select flash mode in sleep mode or in snooze mode. |
| Deep Sleep and Standby Options | | | |
| Deep Sleep and Standby Options > Snooze Options (Not available on every MCU) | | | |
| Snooze Request Source | MCU Specific Options | | Select the event that will enter snooze. |
| Snooze End Sources | MCU Specific Options | | Enable wake from snooze from these sources. |
| DTC state in Snooze Mode | MCU Specific Options | | Enable wake from snooze from this source. |
| Snooze Cancel Source | MCU Specific Options | | Select an interrupt source to cancel snooze. |
| Wake Sources | MCU Specific Options | | Enable wake from deep sleep and standby from these Sources. |

| | | |
|---|---|---|
| Wake Sources 2 | MCU Specific Options | Enable wake from deep sleep and standby from these Sources. |
| RAM Retention Control (Not available on every MCU) | | |
| RAM retention in Standby mode | MCU Specific Options | Select the memory regions that are retained in standby mode. |
| TCM retention in Deep Sleep and Standby modes | MCU Specific Options | Select if Tightly Coupled Memory (TCM) is retained in deep sleep and standby modes. |
| Standby RAM retention in Standby and Deep Standby modes | MCU Specific Options | Select if Standby RAM is retained in standby and deep standby modes. |
| Oscillator LDO Control (Not available on every MCU) | | |
| PLL1 LDO State in standby mode | MCU Specific Options | Select the state PLL1 LDO state in standby mode. |
| PLL2 LDO State in standby mode | MCU Specific Options | Select the state PLL2 LDO state in standby mode. |
| HOCO LDO State in standby mode | MCU Specific Options | Select the state HOCO LDO state in standby mode. |
| Deep Standby Options (Not available on every MCU) | | |
| I/O Port Retention | MCU Specific Options | Select the state of the IO Pins after exiting deep standby mode. |
| Power-Supply Control | MCU Specific Options | Select the state of the internal power supply in deep standby mode. |
| Cancel Sources | MCU Specific Options | Enable wake from deep standby using these sources. |
| Cancel Edges | MCU Specific Options | Falling edge trigger is default. Select sources to enable wake from deep standby with rising edge. |
| Soft Start Mode | MCU Specific Options | The soft start setting affects the inrush current, and wake-up time when waking from |

| | |
|---|---|
| | deep standby mode (See the MCU user manual for more information). |

### Clock Configuration

This module does not have any selectable clock sources.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### Sleep Mode

At power on, by default sleep is set as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the SysTick interrupt used by the RTOS scheduler.

### Deep Sleep Mode

Deep sleep mode is similar to sleep mode with the exception that DTC and DMAC are stopped, access to TCM memory is not available, and only a subset of interrupts are available for waking the CPU.

### Software Standby Mode

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. The contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports are all retained. Software-standby mode allows significant reduction in power consumption, because most of the oscillators are stopped in this mode. Like sleep mode, standby mode requires an interrupt or event be configured and enabled to wake up.

### Snooze Mode

Snooze mode can be used with some MCU peripherals to execute basic tasks while keeping the MCU in a low-power state. Many core peripherals and all clocks can be selected to run during Snooze, allowing for more flexible low-power configuration than Software Standby mode. To enable Snooze, select "Software Standby mode with Snooze mode enabled" for the "Low Power Mode" configuration option. Snooze mode settings (including entry/exit sources) are available under "Standby Options".

### Deep Software Standby Mode

The MCU always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of the wakeup sources configurable in the "Deep Standby

Options" configuration group.

The Reset Status Registers can be used to determine if the reset occurred after coming out of deep software standby. For example, R_SYSTEM->RSTSR0_b.DPSRSTF is set to 1 after a deep software standby reset.

I/O Port Retention can be enabled to maintain I/O port configuration across a deep software standby reset. Retention can be cancelled through the R_LPM_IoKeepClear API.

### Limitations

Developers should be aware of the following limitations when using the LPM:

- Flash stop (code flash disable) is not supported. See the section "Flash Operation Control Register (FLSTOP)" of the RA2/RA4 Family Hardware User's Manual.
- Reduced SRAM retention area in software standby mode is not supported. See the section "Power Save Memory Control Register (PSMCR)" of the RA4 Hardware User's Manual.
- Only one Snooze Request Source can be used at a time.
- When using Snooze mode with SCI0 RXD as the snooze source the system clock must be HOCO and the MOCO, Main Oscillator and PLL clocks must be turned off.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.
- When using the DC-DC regulator (where available), the MCU will temporarily switch to the LDO if Software Standby or Snooze is requested and back again when it is cancelled. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).
- On RA8, there are delays inserted before and after switching to a lower power mode based on BSP configuration options and current CPUCLK speed. These delays do not disable interrupts for the entire delay as required by the RA8 Hardware Manual. Interrupts should be disabled by the application when possible before sleeping to meet the requirements, or the control of interrupts and the delay should be managed by the user to ensure the requirement is fully met.

## Examples

### LPM Sleep Example

This is a basic example of minimal use of the LPM in an application. The LPM instance is opened and the configured low-power mode is entered.

```
void r_lpm_sleep (void)
{
 fsp_err_t err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_sleep);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
```

```
    assert(FSP_SUCCESS == err);
}
```

### LPM Deep Software Standby Example

```
void r_lpm_deep_software_standby (void)
{
 fsp_err_t err;
#ifdef R_SYSTEM_RSTSR0_DPSRSTF_Msk
 /* Check the Deep Software Standby Reset Flag. If it is set, then the MCU is exiting
  * Deep Software Standby mode. */
 if (1U == R_SYSTEM->RSTSR0_b.DPSRSTF)
    {
 /* Clear the IOKEEP bit to allow I/O Port use.
  * Note that this function should be called before opening the LPM driver. */
       err = R_LPM_IoKeepClear(&g_lpm_ctrl);
      assert(FSP_SUCCESS == err);
    }
#endif
    err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Add user code here. */
 /* Reconfigure the module to set the IOKEEP bit before entering deep software
standby. */
    err = R_LPM_LowPowerReconfigure(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
 /* Code after R_LPM_LowPowerModeEnter when using Deep Software Standby never be
executed.
  * Deep software standby exits by resetting the MCU. */
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | lpm_instance_ctrl_t |

## Data Structure Documentation

### ◆ lpm_instance_ctrl_t

| struct lpm_instance_ctrl_t |
|---|
| LPM private control block. DO NOT MODIFY. Initialization occurs when R_LPM_Open() is called. |

## Function Documentation

### ◆ R_LPM_Open()

fsp_err_t R_LPM_Open ( lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg* )

Perform any necessary initialization

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LPM instance opened |
| FSP_ERR_ASSERTION | Null Pointer |
| FSP_ERR_ALREADY_OPEN | LPM instance is already open |
| FSP_ERR_UNSUPPORTED | This MCU does not support Deep Software Standby |
| FSP_ERR_INVALID_ARGUMENT | One of the following:<br>• Invalid snooze entry source<br>• Invalid snooze end sources |
| FSP_ERR_INVALID_MODE | One of the following:<br>• Invalid low power mode<br>• Invalid DTC option for snooze mode<br>• Invalid deep standby end sources<br>• Invalid deep standby end sources edges<br>• Invalid power supply option for deep standby<br>• Invalid IO port option for deep standby<br>• Invalid output port state setting for standby or deep standby<br>• Invalid sources for wake from standby mode<br>• Invalid power supply option for standby<br>• Invalid IO port option for standby<br>• Invalid standby end sources<br>• Invalid standby end sources edges |

### ◆ R_LPM_LowPowerReconfigure()

fsp_err_t R_LPM_LowPowerReconfigure ( lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg* )

Configure a low power mode

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Low power mode successfully applied |
| FSP_ERR_ASSERTION | Null Pointer |
| FSP_ERR_NOT_OPEN | LPM instance is not open |
| FSP_ERR_UNSUPPORTED | This MCU does not support Deep Software Standby |
| FSP_ERR_INVALID_ARGUMENT | One of the following:<br>• Invalid snooze entry source<br>• Invalid snooze end sources |
| FSP_ERR_INVALID_MODE | One of the following:<br>• Invalid low power mode<br>• Invalid DTC option for snooze mode<br>• Invalid deep standby end sources<br>• Invalid deep standby end sources edges<br>• Invalid power supply option for deep standby<br>• Invalid IO port option for deep standby<br>• Invalid output port state setting for standby or deep standby<br>• Invalid sources for wake from standby mode<br>• Invalid power supply option for standby<br>• Invalid IO port option for standby<br>• Invalid standby end sources<br>• Invalid standby end sources edges |

### ◆ R_LPM_LowPowerModeEnter()

fsp_err_t R_LPM_LowPowerModeEnter ( lpm_ctrl_t *const *p_api_ctrl*)

Enter low power mode (sleep/deep sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_ASSERTION | Null pointer. |
| FSP_ERR_NOT_OPEN | LPM instance is not open |
| FSP_ERR_INVALID_MODE | One of the following:<br>• HOCO was not system clock when using snooze mode with SCI0/RXD0.<br>• HOCO was not stable when using snooze mode with SCI0/RXD0.<br>• MOCO was running when using snooze mode with SCI0/RXD0.<br>• MAIN OSCILLATOR was running when using snooze mode with SCI0/RXD0.<br>• PLL was running when using snooze mode with SCI0/RXD0.<br>• Unable to disable ocillator stop detect when using standby or deep standby. |

### ◆ R_LPM_IoKeepClear()

fsp_err_t R_LPM_IoKeepClear ( lpm_ctrl_t *const *p_api_ctrl*)

Clear the IOKEEP bit after deep software standby

**Return values**

| | |
|---|---|
| FSP_SUCCESS | DPSBYCR_b.IOKEEP bit cleared Successfully. |
| FSP_ERR_UNSUPPORTED | Deep standby mode not supported on this MCU. |

◆ **R_LPM_Close()**

| fsp_err_t R_LPM_Close ( lpm_ctrl_t *const *p_api_ctrl*) |
|---|
| Close the LPM Instance<br><br>**Return values**<br>FSP_SUCCESS — LPM driver closed<br>FSP_ERR_NOT_OPEN — LPM instance is not open<br>FSP_ERR_ASSERTION — Null Pointer |

## 5.2.14 RTOS

Modules

### Detailed Description

RTOS Modules.

### Modules

Azure RTOS ThreadX Port (rm_threadx_port)

ThreadX port for RA MCUs.

FreeRTOS Port (rm_freertos_port)

FreeRTOS port for RA MCUs.

### 5.2.14.1 Azure RTOS ThreadX Port (rm_threadx_port)

Modules » RTOS

ThreadX port for RA MCUs.

# Overview

*Note*

***The ThreadX Port does not provide any interfaces to the user. Consult the ThreadX documentation at https://docs.microsoft.com/en-us/azure/rtos/threadx/ for further information.***

### Features

The RA ThreadX port supports the following features:

- Standard ThreadX configurations
- Hardware stack monitor

# Configuration

### Build Time Configurations for ThreadX

The following build time configurations are defined in fsp_cfg/azure/tx/tx_user.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Custom tx_user.h | Manual Entry | | Add a path to your custom tx_user.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Error Checking | • Enabled<br>• Disabled | Enabled | The ThreadX basic API error checking can be bypassed by compiling with the symbol TX_DISABLE_ERROR_CHECKING defined. |
| Max Priorities | Value must be a multiple of 32 and in range 32 to 1024 or empty | 32 | Define the priority levels for ThreadX. Legal values range from 32 to 1024 and MUST be evenly divisible by 32. |
| Minimum Stack | Value must be greater than 0 or empty | 200 | Define the minimum stack for a ThreadX thread on this processor. If the size supplied during thread creation is less than this value, the thread create call will return an error. |
| Stack Filling | • Enabled<br>• Disabled | Enabled | Determine is stack filling is enabled. By default, ThreadX stack filling is enabled, which places an 0xEF pattern in each byte of each |

| | | | |
|---|---|---|---|
| | | | thread's stack. This is used by debuggers with ThreadX-awareness and by the ThreadX run-time stack checking feature. |
| Preemption Threshold | • Enabled<br>• Disabled | Disabled | Determine if preemption-threshold should be disabled. By default, preemption-threshold is disabled. If the application does not use preemption-threshold, it may be disabled to reduce code size and improve performance. |
| Notify Callbacks | • Enabled<br>• Disabled | Disabled | Determine if the notify callback option should be disabled. By default, notify callbacks are disabled. If the application does not use notify callbacks, they may be disabled to reduce code size and improve performance. |
| Inline Thread Resume Suspend | • Enabled<br>• Disabled | Disabled | Determine if the tx_thread_resume and tx_thread_suspend services should have their internal code in-line. This results in a larger image, but improves the performance of the thread resume and suspend services. |
| Not Interruptable | • Enabled<br>• Disabled | Disabled | Determine if the internal ThreadX code is non-interruptable. This results in smaller code size and less processing overhead, but increases the interrupt lockout time. |
| IAR Library Support | • Enabled<br>• Disabled | Disabled | Enable IAR library support (IAR compiler only). When IAR Library Support is Enabled, enable the linker option |

| | | | |
|---|---|---|---|
| | | | --threaded_lib. In the IAR IDE, this can be enabled in Project > Options > General Options > Library Configuration > Enable thread support in library. |
| BSD Support | • Enabled<br>• Disabled | Disabled | Defines TX_THREAD_EXTENSION_1 to bsd_err_no in order to support NXD BSD. |
| FileX Pointer | • Enabled<br>• Disabled | Enabled | Determine if there is a FileX pointer in the thread control block. By default, the pointer is there for legacy/backwards compatibility. The pointer must also be there for applications using FileX. Disable this to save space in the thread control block. |
| Timer | | | |
| Timer Ticks Per Second | Value must be greater than 0 or empty | 100 | Define the number of times the system timer runs per second. Default is 100 ticks per second, which results in a tick every 10ms. |
| Timer Thread Stack Size | Value must be greater than 0 or empty | 1024 | Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled. |
| Timer Thread Priority | Value must be a positive integer or empty | 0 | Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled. |
| Timer Process In ISR | • Enabled<br>• Disabled | Enabled | Determine if timer expirations (application timers, timeouts, and tx_thread_sleep calls should be processed within the a system |

| | | | |
|---|---|---|---|
| | | | timer thread or directly in the timer ISR. When disabled, the timer thread is used. When enabled, timer expiration processing is done directly from the timer ISR, thereby eliminating the timer thread control block, stack, and context switching to activate it. |
| Reactivate Inline | • Enabled<br>• Disabled | Disabled | Determine if in-line timer reactivation should be used within the timer expiration processing. By default, this is disabled and a function call is used. When enabled, reactivating is performed in-line resulting in faster timer processing but slightly larger code size. |
| Timer | • Enabled<br>• Disabled | Enabled | Determine if no timer processing is required. This option will help eliminate the timer processing when not needed. |
| Trace | | | |
| Event Trace | • Enabled<br>• Disabled | Disabled | Determine if the trace event logging code should be enabled. This causes slight increases in code size and overhead, but provides the ability to generate system trace information which is available for viewing in TraceX. |
| Trace Buffer Name | Name must be a valid C symbol | g_tx_trace_buffer | Name of trace buffer symbol, only used if Event Trace is enabled. |
| Memory section for Trace Buffer | Manual Entry | .bss | Specify the memory section where the Trace Buffer will be allocated, only used if Event Trace is enabled. |

| | | | |
|---|---|---|---|
| | | | To view TraceX data, export this buffer as raw binary data to a file (.trx extension recommended) and open it with Azure RTOS TraceX. |
| Trace Buffer Size | Value must be greater than 0 | 65536 | Trace buffer size in bytes, only used if Event Trace is enabled |
| Trace Buffer Number of Registries | Value must be greater than 0 | 30 | Number of registries available to TraceX, only used if Event Trace is enabled |
| Performance | | | |
| Block Pool Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers block pool performance information. |
| Byte Pool Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers byte pool performance information. |
| Event Flags Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers event flags performance information. |
| Mutex Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers mutex performance information. |
| Queue Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers queue performance information. |
| Semaphore Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers semaphore performance information. |
| Thread Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers thread performance information. |
| Timer Performance Info | • Enabled<br>• Disabled | Disabled | When enabled, ThreadX gathers timer performance information. |

| | | |
|---|---|---|
| RA | | |
| Hardware Thread Stack Monitoring | MCU Specific Options | Use RA Hardware Stack Monitors to monitor thread stacks for overflow. Not available on MCUs that support PSPLIM. |
| Interrupts | | |
| SysTick Interrupt Priority | MCU Specific Options | Select the Systick interrupt priority. |
| Maximum Interrupt Priority | MCU Specific Options | The maximum priority (lowest numerical value) an interrupt can have and use scheduler services. Interrupts with higher priority can interrupt most scheduler critical sections. Setting this to Priority 0 (highest) disables this feature. This feature is not available on MCUs that do not have the BASEPRI register. |

### Clock Configuration

The ThreadX port uses the SysTick timer as the system clock. The timer rate is configured in the ThreadX component under General > Timer Ticks Per Second.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Interrupt Priorities

When no threads are ready to run, the ThreadX port spins in the PendSV_Handler, which is fixed at the lowest interrupt priority. The MCU does not service any other interrupts of the lowest priority while no threads are ready to run.

To get around this limitation, the application can create an idle thread that is always ready to run. If the idle thread enters a lower power mode, make sure all interrupts that are required to resume the scheduler can wake the MCU in the configured power mode. If the application expects to wake after a certain number of ticks, the idle thread should not enter standby mode because the SysTick cannot wake the MCU from standby mode. See Low Power Modes (r_lpm) for more information regarding low power modes.

Warning

Do not attempt to wake a thread from an interrupt with the lowest available interrupt

priority unless the application has created an idle thread.

### Hardware Stack Monitor

The hardware stack monitor generates an NMI if the PSP goes out of the memory area for the stack allocated for the current thread. A callback can be registered using R_BSP_GroupIrqWrite() to be called whenever a stack overflow or underflow of the PSP for a particular thread is detected.

### Low Power Modes

The idle processing executes WFI() when no thread is ready to run. If the MCU is configured to enter software standby mode or deep software standby mode when the idle processing executes WFI(), the RA ThreadX port changes the low power mode to sleep mode so the idle processing can wake from SysTick. The low power mode settings are restored when the MCU wakes from sleep mode.

### TrustZone Integration

When using an RTOS in a TrustZone project, Arm recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the thread has allocated a secure context (using tx_thread_secure_stack_allocate).

The secure context can be freed by deleting the thread or calling tx_thread_secure_stack_free.

## Examples

### Stack Monitor Example

This is an example of using the stack monitor in an application.

```
#if BSP_FEATURE_BSP_HAS_SP_MON
void stack_monitor_callback(bsp_grp_irq_t irq);
void stack_monitor_callback (bsp_grp_irq_t irq)
{
 FSP_PARAMETER_NOT_USED(irq);
 if (1U == R_MPU_SPMON->SP[0].CTL_b.ERROR)
    {
 /* Handle main stack monitor error here. */
    }
 if (1U == R_MPU_SPMON->SP[1].CTL_b.ERROR)
    {
 /* Handle process stack monitor error here. */
    }
}
void rm_threadx_port_stack_monitor_example (void)
{
```

```
 /* Register a callback to be called when the stack goes outside the allocated stack
area. */
 R_BSP_GroupIrqWrite(BSP_GRP_IRQ_MPU_STACK, stack_monitor_callback);
}
#else
/* Allocate stack space to return from UsageFault. */
uint32_t g_stack_overflow_exception_stack[8] BSP_ALIGN_VARIABLE(BSP_STACK_ALIGNMENT)
BSP_PLACE_IN_SECTION(
    BSP_UNINIT_SECTION_PREFIX ".ram_noinit");
/* MCUs that do not have an SPMON stack monitor use PSPLIM to detect stack overflows.
When a stack overflow error
 * occurs, the UsageFault_Handler fires if it has been enabled. */
void UsageFault_Handler (void)
{
 register uint32_t cfsr = SCB->CFSR;
 if (cfsr & SCB_CFSR_STKOF_Msk)
    {
 /* Update PSP and PSPLIM to point to an exception stack frame allocated for stack
overflows. */
 register uint32_t * p_exception_stack_frame = (uint32_t *)
(&g_stack_overflow_exception_stack);
       __set_PSP((uint32_t) p_exception_stack_frame);
       __set_PSPLIM((uint32_t) p_exception_stack_frame);
 /* Clear XPSR, only set T-bit. */
       p_exception_stack_frame[7] = 1U << 24;
 /* Set PC to stack overflow error while loop. When execution returns from the
UsageFault, it will go to the
  * stack_overflow_error_occurred function. It cannot return to the location where
the fault occurred because
  * the MCU does not save the exception stack frame to the stack when a stack
overflow error occurs. */
       p_exception_stack_frame[6] = (uint32_t) stack_overflow_error_occurred;
    }
 /* Clear flags. */
```

```
    SCB->CFSR = cfsr;
}
/* This function is called from UsageFault_Handler after a stack overflow occurs. */
void stack_overflow_error_occurred (void)
{
 /* When recovering from a stack overflow, move the thread to a while(1) loop. */
 while (1)
    {
 /* Do nothing. */
    }
}
void rm_threadx_port_stack_monitor_example (void)
{
 /* Enable usage fault. */
    SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk;
}
#endif
```

### TrustZone Example

This is an example of calling tx_thread_secure_stack_allocate before calling any non-secure callable functions in a thread.

```
extern TX_THREAD * _tx_thread_current_ptr;
void rm_threadx_port_trustzone_thread_example (void)
{
 /* When ThreadX is used in a non-secure TrustZone application,
tx_thread_secure_stack_allocate must be called prior
  * to calling any non-secure callable function in a thread. The first parameter is a
pointer to the thread control block.
  * This function can be called when the thread is created or in the thread before an
non-secure callable function is
  * called. The second parameter is unused in the FSP implementation. */
    UINT status = tx_thread_secure_stack_allocate(_tx_thread_current_ptr, 0);
    assert(TX_SUCCESS == status);
```

```
    rm_threadx_port_nsc_function();
}
```

### 5.2.14.2 FreeRTOS Port (rm_freertos_port)

Modules » RTOS

FreeRTOS port for RA MCUs.

# Overview

*Note*

***The FreeRTOS Port does not provide any interfaces to the user. Consult the FreeRTOS documentation at https://www.freertos.org/Documentation for further information.***

#### Features

The RA FreeRTOS port supports the following features:

- Standard FreeRTOS configurations
- Hardware stack monitor

# Configuration

*Note*

***The FreeRTOS Port and libraries use printf by default when logging is enabled. printf requires a heap (BSP Tab -> Properties -> RA Common -> Heap size (bytes)).***

#### Build Time Configurations for all

The following build time configurations are defined in aws/FreeRTOSConfig.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Custom FreeRTOSConfig.h | Manual Entry | | Add a path to your custom FreeRTOSConfig.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Use Preemption | • Enabled<br>• Disabled | Enabled | Set to Enabled to use the preemptive RTOS scheduler, or Disabled |

| | | | |
|---|---|---|---|
| | | | to use the cooperative RTOS scheduler. |
| Use Port Optimised Task Selection | • Enabled<br>• Disabled | Disabled | Some FreeRTOS ports have two methods of selecting the next task to execute - a generic method, and a method that is specific to that port.<br><br>The Generic method:<br>Is used when Use Port Optimized Task Selection is set to 0, or when a port specific method is not implemented.<br>Can be used with all FreeRTOS ports.<br>Is completely written in C, making it less efficient than a port specific method.<br>Does not impose a limit on the maximum number of available priorities.<br><br>A port specific method:<br>Is not available for all ports.<br>Is used when Use Port Optimized Task Selection is Enabled.<br>Relies on one or more architecture specific assembly instructions (typically a Count Leading Zeros [CLZ] or equivalent instruction) so can only be used with the architecture for which it was specifically written.<br>Is more efficient than the generic method.<br>Typically imposes a limit of 32 on the maximum number of available priorities. |
| Use Tickless Idle | • Enabled<br>• Disabled | Disabled | Set Use Tickless Idle to Enabled to use the low power tickless mode, or Disabled to keep the |

| | | | |
|---|---|---|---|
| | | | tick interrupt running at all times. Low power tickless implementations are not provided for all FreeRTOS ports. |
| Cpu Clock Hz | Manual Entry | SystemCoreClock | Enter the frequency in Hz at which the internal clock that drives the peripheral used to generate the tick interrupt will be executing - this is normally the same clock that drives the internal CPU clock. This value is required in order to correctly configure timer peripherals. |
| Tick Rate Hz | Value must be greater than 0 | 1000 | The frequency of the RTOS tick interrupt. The tick interrupt is used to measure time. Therefore a higher tick frequency means time can be measured to a higher resolution. However, a high tick frequency also means that the RTOS kernel will use more CPU time so be less efficient. The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required.<br><br>More than one task can share the same priority. The RTOS scheduler will share processor time between tasks of the same priority by switching between the tasks during each RTOS tick. A high tick rate frequency will therefore also have the effect of reducing the |

| | | | |
|---|---|---|---|
| | | | 'time slice' given to each task. |
| Max Priorities | Must be an integer and greater than 0 | 5 | The number of priorities available to the application tasks. Any number of tasks can share the same priority.<br>Each available priority consumes RAM within the RTOS kernel so this value should not be set any higher than actually required by your application. |
| Minimal Stack Size | Must be an integer and greater than 0 | 128 | The size of the stack used by the idle task. Generally this should not be reduced from the value set in the FreeRTOSConfig.h file provided with the demo application for the port you are using. Like the stack size parameter to the xTaskCreate() and xTaskCreateStatic() functions, the stack size is specified in words, not bytes. If each item placed on the stack is 32-bits, then a stack size of 100 means 400 bytes (each 32-bit stack item consuming 4 bytes). |
| Max Task Name Len | Must be an integer and greater than 0 | 16 | The maximum permissible length of the descriptive name given to a task when the task is created. The length is specified in the number of characters including the NULL termination byte. |
| Use 16-bit Ticks | Disabled | Disabled | Time is measured in 'ticks' - which is the number of times the tick interrupt has executed since the |

| | | | |
|---|---|---|---|
| | | | RTOS kernel was started. The tick count is held in a variable of type TickType_t. Defining configUSE_16_BIT_TICKS as 1 causes TickType_t to be defined (typedef'ed) as an unsigned 16bit type. Defining configUSE_16_BIT_TICKS as 0 causes TickType_t to be defined (typedef'ed) as an unsigned 32bit type.<br><br>Using a 16-bit type will greatly improve performance on 8- and 16-bit architectures, but limits the maximum specifiable time period to 65535 'ticks'. Therefore, assuming a tick frequency of 250Hz, the maximum time a task can delay or block when a 16bit counter is used is 262 seconds, compared to 17179869 seconds when using a 32-bit counter. |
| Idle Should Yield | • Enabled<br>• Disabled | Enabled | This parameter controls the behaviour of tasks at the idle priority. It only has an effect if:<br>The preemptive scheduler is being used.<br>The application creates tasks that run at the idle priority.<br>If Use Time Slicing is Enabled then tasks that share the same priority will time slice. If none of the tasks get preempted then it might be assumed that each task at a given priority will be |

allocated an equal amount of processing time - and if the priority is above the idle priority then this is indeed the case. When tasks share the idle priority the behaviour can be slightly different. If Idle Should Yield is Enabled then the idle task will yield immediately if any other task at the idle priority is ready to run. This ensures the minimum amount of time is spent in the idle task when application tasks are available for scheduling. This behaviour can however have undesirable effects (depending on the needs of your application) as depicted below:

The diagram above shows the execution pattern of four tasks that are all running at the idle priority. Tasks A, B and C are application tasks. Task I is the idle task. A context switch occurs with regular period at times T0, T1, ..., T6. When the idle task yields task A starts to execute - but the idle task has already consumed some of the current time slice. This results in task I and task A effectively sharing the same time slice. The application tasks B and C therefore get more processing time than the application task A.

| | | | |
|---|---|---|---|
| | | | This situation can be avoided by:<br><br>If appropriate, using an idle hook in place of separate tasks at the idle priority.<br>Creating all application tasks at a priority greater than the idle priority.<br>Setting Idle Should Yield to Disabled.<br>Setting Idle Should Yield to Disabled prevents the idle task from yielding processing time until the end of its time slice. This ensure all tasks at the idle priority are allocated an equal amount of processing time (if none of the tasks get pre-empted) - but at the cost of a greater proportion of the total processing time being allocated to the idle task. |
| Use Task Notifications | • Enabled<br>• Disabled | Enabled | Setting Use Task Notifications to Enabled will include direct to task notification functionality and its associated API in the build.<br>Setting Use Task Notifications to Disabled will exclude direct to task notification functionality and its associated API from the build.<br><br>Each task consumes 8 additional bytes of RAM when direct to task notifications are included in the build. |
| Task Notification Array Entries | Must be an integer and greater than 0 | 1 | Set the Task Notifications Array |

| | | | |
|---|---|---|---|
| | | | Entries to enter the size of array needed |
| Use Mutexes | • Enabled<br>• Disabled | Disabled | Set to Enabled to include mutex functionality in the build, or Disabled to omit mutex functionality from the build. Readers should familiarise themselves with the differences between mutexes and binary semaphores in relation to the FreeRTOS functionality. |
| Use Recursive Mutexes | • Enabled<br>• Disabled | Disabled | Set to Enabled to include recursive mutex functionality in the build, or Disabled to omit recursive mutex functionality from the build. |
| Use Counting Semaphores | • Enabled<br>• Disabled | Enabled | Set to Enabled to include counting semaphore functionality in the build, or Disabled to omit counting semaphore functionality from the build. |
| Queue Registry Size | Value must be positive integer greater than or equal to 0 | 10 | The queue registry has two purposes, both of which are associated with RTOS kernel aware debugging:<br>It allows a textual name to be associated with a queue for easy queue identification within a debugging GUI.<br>It contains the information required by a debugger to locate each registered queue and semaphore.<br>The queue registry has no purpose unless you are using a RTOS kernel aware debugger. Registry |

| | | | |
|---|---|---|---|
| | | | Size defines the maximum number of queues and semaphores that can be registered. Only those queues and semaphores that you want to view using a RTOS kernel aware debugger need be registered. See the API reference documentation for vQueueAddToRegistry( ) and vQueueUnregisterQueue() for more information. |
| Use Queue Sets | • Enabled<br>• Disabled | Disabled | Set to Enabled to include queue set functionality (the ability to block, or pend, on multiple queues and semaphores), or Disabled to omit queue set functionality. |
| Use Time Slicing | • Enabled<br>• Disabled | Disabled | If Use Time Slicing is Enabled, FreeRTOS uses prioritised preemptive scheduling with time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick interrupt. If Use Time Slicing is Disabled then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between tasks of equal priority just because a tick interrupt has occurred. |
| Use Newlib Reentrant | • Enabled<br>• Disabled | Disabled | If Use Newlib Reentrant is Enabled then a newlib reent structure will be allocated for |

| | | | |
|---|---|---|---|
| | | | each created task. Note Newlib support has been included by popular demand, but is not used by the FreeRTOS maintainers themselves. FreeRTOS is not responsible for resulting newlib operation. User must be familiar with newlib and must provide system-wide implementations of the necessary stubs. Be warned that (at the time of writing) the current newlib design implements a system-wide malloc() that must be provided with locks. |
| Enable Backward Compatibility | • Enabled<br>• Disabled | Disabled | The FreeRTOS.h header file includes a set of #define macros that map the names of data types used in versions of FreeRTOS prior to version 8.0.0 to the names used in FreeRTOS version 8.0.0. The macros allow application code to update the version of FreeRTOS they are built against from a pre 8.0.0 version to a post 8.0.0 version without modification. Setting Enable Backward Compatibility to Disabled in FreeRTOSConfig.h excludes the macros from the build, and in so doing allowing validation that no pre version 8.0.0 names are being used. |
| Num Thread Local Storage Pointers | Value must be positive integer greater than or equal to 0 | 5 | Sets the number of indexes in each task's thread local storage array. |
| Stack Depth Type | Manual Entry | uint32_t | Sets the type used to |

| | | | |
|---|---|---|---|
| | | | specify the stack depth in calls to xTaskCreate(), and various other places stack sizes are used (for example, when returning the stack high water mark). Older versions of FreeRTOS specified stack sizes using variables of type UBaseType_t, but that was found to be too restrictive on 8-bit microcontrollers. Stack Depth Type removes that restriction by enabling application developers to specify the type to use. |
| Message Buffer Length Type | Manual Entry | size_t | FreeRTOS Message buffers use variables of type Message Buffer Length Type to store the length of each message. If Message Buffer Length Type is not defined then it will default to size_t. If the messages stored in a message buffer will never be larger than 255 bytes then defining Message Buffer Length Type to uint8_t will save 3 bytes per message on a 32-bit microcontroller. Likewise if the messages stored in a message buffer will never be larger than 65535 bytes then defining Message Buffer Length Type to uint16_t will save 2 bytes per message on a 32-bit microcontroller. |
| Library Max Syscall Interrupt Priority | MCU Specific Options | | The highest interrupt priority that can be used by any interrupt service routine that |

makes calls to interrupt safe FreeRTOS API functions. DO NOT CALL INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER PRIORITY THAN THIS! (higher priorities are lower numeric values)

Below is explanation for macros that are set based on this value from FreeRTOS website.

In the RA port, configKERNEL_INTERRUPT_PRIORITY is not used and the kernel runs at the lowest priority.

Note in the following discussion that only API functions that end in "FromISR" can be called from within an interrupt service routine.

configMAX_SYSCALL_INTERRUPT_PRIORITY sets the highest interrupt priority from which interrupt safe FreeRTOS API functions can be called.

A full interrupt nesting model is achieved by setting configMAX_SYSCALL_INTERRUPT_PRIORITY above (that is, at a higher priority level) than configKERNEL_INTERRUPT_PRIORITY. This means the FreeRTOS kernel does not completely disable interrupts, even inside critical sections. Further, this is achieved without the

disadvantages of a segmented kernel architecture.

Interrupts that do not call API functions can execute at priorities above configMAX_SYSCALL_INTERRUPT_PRIORITY and therefore never be delayed by the RTOS kernel execution.

A special note for Arm Cortex-M users: Please read the page dedicated to interrupt priority settings on Arm Cortex-M devices. As a minimum, remember that Arm Cortex-M cores use numerically low priority numbers to represent HIGH priority interrupts, which can seem counter-intuitive and is easy to forget! If you wish to assign an interrupt a low priority do NOT assign it a priority of 0 (or other low numeric value) as this can result in the interrupt actually having the highest priority in the system - and therefore potentially make your system crash if this priority is above config MAX_SYSCALL_INTERRUPT_PRIORITY.

The lowest priority on a Arm Cortex-M core is in fact 255 - however different Arm Cortex-M vendors implement a different number of priority bits and supply library functions that expect priorities to be specified in different ways. For example, on the RA6M3 the lowest

| | | | |
|---|---|---|---|
| | | | priority you can specify is 15 - and the highest priority you can specify is 0. |
| Assert | Manual Entry | assert(x) | The semantics of the configASSERT() macro are the same as the standard C assert() macro. An assertion is triggered if the parameter passed into configASSERT() is zero. configASSERT() is called throughout the FreeRTOS source files to check how the application is using FreeRTOS. It is highly recommended to develop FreeRTOS applications with configASSERT() defined.<br><br>The example definition (shown at the top of the file and replicated below) calls vAssertCalled(), passing in the file name and line number of the triggering configASSERT() call (__FILE__ and __LINE__ are standard macros provided by most compilers). This is just for demonstration as vAssertCalled() is not a FreeRTOS function, configASSERT() can be defined to take whatever action the application writer deems appropriate.<br><br>It is normal to define configASSERT() in such a way that it will prevent the application from executing any further. This if for two reasons; stopping the application at the point |

of the assertion allows the cause of the assertion to be debugged, and executing past a triggered assertion will probably result in a crash anyway.

Note defining configASSERT() will increase both the application code size and execution time. When the application is stable the additional overhead can be removed by simply commenting out the configASSERT() definition in FreeRTOSConfig.h.

/* Define configASSERT() to call vAssertCalled() if the assertion fails. The assertion
has failed if the value of the parameter passed into configASSERT() equals zero. */
#define configASSERT( ( x ) ) if( ( x ) == 0 ) vAssertCalled( __FILE__, __LINE__ )
If running FreeRTOS under the control of a debugger, then configASSERT() can be defined to just disable interrupts and sit in a loop, as demonstrated below. That will have the effect of stopping the code on the line that failed the assert test - pausing the debugger will then immediately take you to the offending line so you can see why it failed.

| | | | |
|---|---|---|---|
| | | | /* Define configASSERT() to disable interrupts and sit in a loop. */ #define configASSERT( ( x ) ) if( ( x ) == 0 ) { taskDISABLE_INTERRUPTS(); for( ;; ); } |
| Include Application Defined Privileged Functions | • Enabled<br>• Disabled | Disabled | Include Application Defined Privileged Functions is only used by FreeRTOS MPU. If Include Application Defined Privileged Functions is Enabled then the application writer must provide a header file called "application_defined_privileged_functions.h", in which functions the application writer needs to execute in privileged mode can be implemented. Note that, despite having a .h extension, the header file should contain the implementation of the C functions, not just the functions' prototypes.<br><br>Functions implemented in "application_defined_privileged_functions.h" must save and restore the processor's privilege state using the prvRaisePrivilege() function and portRESET_PRIVILEGE() macro respectively. For example, if a library provided print function accesses RAM that is outside of the control of the application writer, and therefore cannot be allocated to a memory protected user mode task, then the print function can |

| | | | |
|---|---|---|---|
| | | | be encapsulated in a privileged function using the following code:<br><br>void MPU_debug_printf( const char *pcMessage )<br>{<br>/* State the privilege level of the processor when the function was called. */<br>BaseType_t xRunningPrivileged = prvRaisePrivilege();<br><br>/* Call the library function, which now has access to all RAM. */<br>debug_printf( pcMessage );<br><br>/* Reset the processor privilege level to its original value. */<br>portRESET_PRIVILEGE( xRunningPrivileged );<br>}<br>This technique should only be use during development, and not deployment, as it circumvents the memory protection. |
| Hooks | | | |
| Use Idle Hook | • Enabled<br>• Disabled | Enabled | Set to Enabled if you wish to use an idle hook, or Disabled to omit an idle hook. |
| Use Malloc Failed Hook | • Enabled<br>• Disabled | Disabled | The kernel uses a call to pvPortMalloc() to allocate memory from the heap each time a task, queue or semaphore is created. The official FreeRTOS download includes four sample memory allocation schemes for this purpose. The schemes are |

| | | | |
|---|---|---|---|
| | | | implemented in the heap_1.c, heap_2.c, heap_3.c, heap_4.c and heap_5.c source files respectively. Use Malloc Failed Hook is only relevant when one of these three sample schemes is being used. The malloc() failed hook function is a hook (or callback) function that, if defined and configured, will be called if pvPortMalloc() ever returns NULL. NULL will be returned only if there is insufficient FreeRTOS heap memory remaining for the requested allocation to succeed.<br><br>If Use Malloc Failed Hook is Enabled then the application must define a malloc() failed hook function. If Use Malloc Failed Hook is set to Dosab;ed then the malloc() failed hook function will not be called, even if one is defined. Malloc() failed hook functions must have the name and prototype shown below.<br><br>void vApplicationMalloc FailedHook( void ); |
| Use Daemon Task Startup Hook | • Enabled<br>• Disabled | Disabled | If Use Timers and Use Daemon Task Startup Hook are both Enabled then the application must define a hook function that has the exact name and prototype as shown below. The hook function will be called exactly once when the RTOS daemon task |

| | | | |
|---|---|---|---|
| | | | (also known as the timer service task) executes for the first time. Any application initialisation code that needs the RTOS to be running can be placed in the hook function. void void vApplicationDaemonTaskStartupHook( void ); |
| Use Tick Hook | • Enabled<br>• Disabled | Disabled | Set to Enabled if you wish to use an tick hook, or Disabled to omit an tick hook. |
| Check For Stack Overflow | • Enabled<br>• Disabled | Disabled | The stack overflow detection page describes the use of this parameter. This is not recommended for RA MCUs with hardware stack monitor support. RA MCU designs should enable the RA hardware stack monitor instead. |
| Stats | | | |
| Use Trace Facility | • Enabled<br>• Disabled | Disabled | Set to Enabled if you wish to include additional structure members and functions to assist with execution visualisation and tracing. |
| Use Stats Formatting Functions | • Enabled<br>• Disabled | Disabled | Set Use Trace Facility and Use Stats Formatting Functions to Enabled to include the vTaskList() and vTaskGetRunTimeStats() functions in the build. Setting either to Disabled will omit vTaskList() and vTaskGetRunTimeStates() from the build. |
| Generate Run Time Stats | • Enabled<br>• Disabled | Disabled | The Run Time Stats page describes the use of this parameter. |
| Memory Allocation | | | |

| | | | |
|---|---|---|---|
| Clear Memory on Free | • Enabled<br>• Disabled | Disabled | If set to 1, then blocks of memory allocated using pvPortMalloc() will be cleared when freed using vPortFree() |
| Support Static Allocation | • Enabled<br>• Disabled | Enabled | If Support Static Allocation is Enabled then RTOS objects can be created using RAM provided by the application writer.<br>If Support Static Allocation is Disabled then RTOS objects can only be created using RAM allocated from the FreeRTOS heap.<br><br>If Support Static Allocation is left undefined it will default to 0.<br><br>If Support Static Allocation is Enabled then the application writer must also provide two callback functions: vApplication GetIdleTaskMemory() to provide the memory for use by the RTOS Idle task, and (if Use Timers is Enabled) vAp plicationGetTimerTask Memory() to provide memory for use by the RTOS Daemon/Timer Service task. Examples are provided below.<br><br>/* Support Static Allocation is Enabled, so the application must provide an implementation of vAp plicationGetIdleTaskMe mory() to provide the memory that is used by the Idle task. */ void vApplicationGetIdl eTaskMemory( StaticTask_t **ppxIdleT |

```
askTCBBuffer,<br>
StackType_t **ppxIdleT
askStackBuffer,<br>
uint32_t
*pulIdleTaskStackSize )
{
/* If the buffers to be
provided to the Idle
task are declared
inside this
function then they
must be declared static
- otherwise they will be
allocated on
the stack and so not
exists after this
function exits. */
static StaticTask_t
xIdleTaskTCB;
static StackType_t
uxIdleTaskStack[ config
MINIMAL_STACK_SIZE ];

/* Pass out a pointer to
the StaticTask_t
structure in which the
Idle task's
state will be stored. */
*ppxIdleTaskTCBBuffer
=

/* Pass out the array
that will be used as the
Idle task's stack. */
*ppxIdleTaskStackBuffe
r = uxIdleTaskStack;

/* Pass out the size of
the array pointed to by
*ppxIdleTaskStackBuffe
r.
Note that, as the array
is necessarily of type
StackType_t,
configMINIMAL_STACK_
SIZE is specified in
words, not bytes. */
*pulIdleTaskStackSize
= configMINIMAL_STAC
K_SIZE;
}
/*-----------------------------
-----------------------------*/

/* Support Static
```

Allocation and Use Timers are both Enabled, so the application must provide an implementation of vApplicationGetTimerTaskMemory()
to provide the memory that is used by the Timer service task. */
void vApplicationGetTimerTaskMemory(
StaticTask_t **ppxTimerTaskTCBBuffer,<br>
StackType_t **ppxTimerTaskStackBuffer,<br>
uint32_t *pulTimerTaskStackSize )
{
/* If the buffers to be provided to the Timer task are declared inside this
function then they must be declared static - otherwise they will be allocated on
the stack and so not exists after this function exits. */
static StaticTask_t xTimerTaskTCB;
static StackType_t uxTimerTaskStack[ configTIMER_TASK_STACK_DEPTH ];

/* Pass out a pointer to the StaticTask_t structure in which the Timer
task's state will be stored. */
*ppxTimerTaskTCBBuffer =

/* Pass out the array that will be used as the Timer task's stack. */
*ppxTimerTaskStackBuffer =
uxTimerTaskStack;

| | | | |
|---|---|---|---|
| | | | /* Pass out the size of the array pointed to by *ppxTimerTaskStackBuffer.<br>Note that, as the array is necessarily of type StackType_t, configTIMER_TASK_STACK_DEPTH is specified in words, not bytes. */<br>*pulTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH;<br>}<br><br>Examples of the callback functions that must be provided by the application to supply the RAM used by the Idle and Timer Service tasks if Support Static Allocation is Enabled.<br><br>See the Static Vs Dynamic Memory Allocation page for more information. |
| Support Dynamic Allocation | • Enabled<br>• Disabled | Disabled | If Support Dynamic Allocation is Enabled then RTOS objects can be created using RAM that is automatically allocated from the FreeRTOS heap.<br>If Support Dynamic Allocation is set to 0 then RTOS objects can only be created using RAM provided by the application writer.<br><br>See the Static Vs Dynamic Memory Allocation page for more information. |
| Total Heap Size | Value must be positive integer greater than or equal to 0 | 1024 | The total amount of RAM available in the FreeRTOS heap.<br>This value will only be used if Support Dynamic Allocation is Enabled and the |

| | | | |
|---|---|---|---|
| | | | application makes use of one of the sample memory allocation schemes provided in the FreeRTOS source code download. See the memory configuration section for further details. |
| Application Allocated Heap | • Enabled<br>• Disabled | Disabled | By default the FreeRTOS heap is declared by FreeRTOS and placed in memory by the linker. Setting Application Allocated Heap to Enabled allows the heap to instead be declared by the application writer, which allows the application writer to place the heap wherever they like in memory.<br>If heap_1.c, heap_2.c or heap_4.c is used, and Application Allocated Heap is Enabled, then the application writer must provide a uint8_t array with the exact name and dimension as shown below. The array will be used as the FreeRTOS heap. How the array is placed at a specific memory location is dependent on the compiler being used - refer to your compiler's documentation.<br><br>uint8_t ucHeap[ configTOTAL_HEAP_SIZE ]; |
| Timers | | | |
| Use Timers | • Enabled<br>• Disabled | Enabled | Set to Enabled to include software timer functionality, or Disabled to omit software timer functionality. See the |

| | | | |
|---|---|---|---|
| | | | FreeRTOS software timers page for a full description. |
| Timer Task Priority | Value must be positive integer greater than or equal to 0 | 3 | Sets the priority of the software timer service/daemon task. See the FreeRTOS software timers page for a full description. |
| Timer Queue Length | Value must be a non-negative integer | 10 | Sets the length of the software timer command queue. See the FreeRTOS software timers page for a full description. |
| Timer Task Stack Depth | Value must be a non-negative integer | 128 | Sets the stack depth allocated to the software timer service/daemon task. See the FreeRTOS software timers page for a full description. |
| Optional Functions | | | |
| vTaskPrioritySet() Function | • Enabled<br>• Disabled | Enabled | Include vTaskPrioritySet() function in build |
| uxTaskPriorityGet() Function | • Enabled<br>• Disabled | Enabled | Include uxTaskPriorityGet() function in build |
| vTaskDelete() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelete() function in build |
| vTaskSuspend() Function | • Enabled<br>• Disabled | Enabled | Include vTaskSuspend() function in build |
| xResumeFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xResumeFromISR() function in build |
| vTaskDelayUntil() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelayUntil() function in build |
| vTaskDelay() Function | • Enabled<br>• Disabled | Enabled | Include vTaskDelay() function in build |
| xTaskGetSchedulerState() Function | • Enabled<br>• Disabled | Enabled | Include xTaskGetSchedulerState() function in build |
| xTaskGetCurrentTaskHandle() Function | • Enabled<br>• Disabled | Enabled | Include xTaskGetCurrentTaskHandle() |

| | | | |
|---|---|---|---|
| | | | function in build |
| uxTaskGetStackHighWaterMark() Function | • Enabled<br>• Disabled | Disabled | Include uxTaskGetStackHighWaterMark() function in build |
| xTaskGetIdleTaskHandle() Function | • Enabled<br>• Disabled | Disabled | Include xTaskGetIdleTaskHandle() function in build |
| eTaskGetState() Function | • Enabled<br>• Disabled | Disabled | Include eTaskGetState() function in build |
| xEventGroupSetBitFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xEventGroupSetBitFromISR() function in build |
| xTimerPendFunctionCall() Function | • Enabled<br>• Disabled | Disabled | Include xTimerPendFunctionCall() function in build |
| xTaskAbortDelay() Function | • Enabled<br>• Disabled | Disabled | Include xTaskAbortDelay() function in build |
| xTaskGetHandle() Function | • Enabled<br>• Disabled | Disabled | Include xTaskGetHandle() function in build |
| xTaskResumeFromISR() Function | • Enabled<br>• Disabled | Enabled | Include xTaskResumeFromISR() function in build |
| RA | | | |
| Hardware Stack Monitor | • Enabled<br>• Disabled | Disabled | Include RA stack monitor |
| Logging | | | |
| Print String Function | Manual Entry | vLoggingPrint(x) | |
| Logging Include Time and Task Name | • Disabled<br>• Enabled | Disabled | |
| Debug Logging Name | Manual Entry | Log Name | |
| Logging Level | • LOG_NONE<br>• LOG_ERROR<br>• LOG_WARN<br>• LOG_INFO<br>• LOG_DEBUG | LOG_NONE | Set the logging level |

### Clock Configuration

The FreeRTOS port uses the SysTick timer as the system clock. The timer rate is configured in the FreeRTOS component under General > Tick Rate Hz.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Hardware Stack Monitor (PSPLIM)

A UsageFault is generated if PSP goes out of the memory area for the stack allocated for the current task. If UsageFault is not enabled, it is escalated to HardFault.

### Hardware Stack Monitor (SPMON)

The hardware stack monitor generates an NMI if the PSP goes out of the memory area for the stack allocated for the current task. A callback can be registered using R_BSP_GroupIrqWrite() to be called whenever a stack overflow or underflow of the PSP for a particular thread is detected.

### Stack Monitor Underflow Detection

By default the hardware stack monitor only checks for overflow of the process stack. To check for underflow define configRECORD_STACK_HIGH_ADDRESS as 1 on the command line.

### Low Power Modes

When FreeRTOS is configured to use tickless idle, the idle task executes WFI() when no task is ready to run. If the MCU is configured to enter software standby mode or deep software standby mode when the idle task executes WFI(), the RA FreeRTOS port changes the low power mode to sleep mode so the idle task can wake from SysTick. The low power mode settings are restored when the MCU wakes from sleep mode.

### TrustZone Integration

When using an RTOS in a TrustZone project, Arm recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the task has allocated a secure context (using portALLOCATE_SECURE_CONTEXT).

The secure context can be freed by deleting the thread or using the portCLEAN_UP_TCB(pxTCB) macro.

# Examples

### Stack Monitor Example

This is an example of using the stack monitor in an application.

```
#if BSP_FEATURE_BSP_HAS_SP_MON
void stack_monitor_callback (bsp_grp_irq_t irq)
{
 FSP_PARAMETER_NOT_USED(irq);
 if (1U == R_MPU_SPMON->SP[0].CTL_b.ERROR)
    {
 /* Handle main stack monitor error here. */
```

```
    }
 if (1U == R_MPU_SPMON->SP[1].CTL_b.ERROR)
    {
 /* Handle process stack monitor error here. */
    }
}
void rm_freertos_port_stack_monitor_example (void)
{
 /* Register a callback to be called when the stack goes outside the allocated stack
area. */
 R_BSP_GroupIrqWrite(BSP_GRP_IRQ_MPU_STACK, stack_monitor_callback);
}
#else
/* Allocate stack space to return from UsageFault. */
uint32_t g_stack_overflow_exception_stack[8] BSP_ALIGN_VARIABLE(BSP_STACK_ALIGNMENT)
BSP_PLACE_IN_SECTION(
    BSP_UNINIT_SECTION_PREFIX ".ram_noinit");
/* MCUs that do not have an SPMON stack monitor use PSPLIM to detect stack overflows.
When a stack overflow error
 * occurs, the UsageFault_Handler fires if it has been enabled. */
void UsageFault_Handler (void)
{
 register uint32_t cfsr = SCB->CFSR;
 if (cfsr & SCB_CFSR_STKOF_Msk)
    {
 /* Update PSP and PSPLIM to point to an exception stack frame allocated for stack
overflows. */
 register uint32_t * p_exception_stack_frame = (uint32_t *)
(&g_stack_overflow_exception_stack);
       __set_PSP((uint32_t) p_exception_stack_frame);
       __set_PSPLIM((uint32_t) p_exception_stack_frame);
 /* Clear XPSR, only set T-bit. */
       p_exception_stack_frame[7] = 1U << 24;
 /* Set PC to stack overflow error while loop. When execution returns from the
```

```
UsageFault, it will go to the
  * stack_overflow_error_occurred function. It cannot return to the location where
the fault occurred because
  * the MCU does not save the exception stack frame to the stack when a stack
overflow error occurs. */
       p_exception_stack_frame[6] = (uint32_t) stack_overflow_error_occurred;
    }
 /* Clear flags. */
    SCB->CFSR = cfsr;
}
/* This function is called from UsageFault_Handler after a stack overflow occurs. */
void stack_overflow_error_occurred (void)
{
 /* When recovering from a stack overflow, move the task to a while(1) loop. */
 while (1)
    {
 /* Do nothing. */
    }
}
void rm_freertos_port_stack_monitor_example (void)
{
 /* Enable usage fault. */
    SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk;
}
#endif
```

### TrustZone Example

This is an example of calling portALLOCATE_SECURE_CONTEXT before calling any non-secure callable functions in a task.

```
void rm_freertos_port_trustzone_task_example (void)
{
 /* When FreeRTOS is used in a non-secure TrustZone application,
portALLOCATE_SECURE_CONTEXT must be called prior
```

```
  * to calling any non-secure callable function in a task. The parameter is unused in
the FSP implementation. */
    portALLOCATE_SECURE_CONTEXT(0);
    rm_freertos_port_nsc_function();
}
```

## 5.2.15 Security

Modules

### Detailed Description

Security Modules.

### Modules

| | |
|---|---|
| | AWS Device Provisioning<br>AWS Device Provisioning example software. |
| | Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto)<br>Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API. |
| | Mbed Crypto H/W Acceleration (rm_psa_crypto)<br>Hardware acceleration for the mbedCrypto implementation of the Arm PSA Crypto API. |
| | Ocrypto (rm_ocrypto_port)<br>Ocrypto software only crypto on RA2E1. |
| | Renesas Secure IP (r_rsip_e11a_protected_cavp)<br>Driver for the CAVP Certified Renesas Secure IP (RSIP-E11A) on RA MPUs. |
| | Renesas Secure IP (r_rsip_e51a_protected_cavp)<br>Driver for the CAVP Certified Renesas Secure IP (RSIP-E51A) on RA MPUs. |

| | |
|---|---|
| | Renesas Secure IP (r_rsip_protected)<br>Driver for the Renesas Secure IP on RA MPUs. This module implements the RSIP Interface. |
| | SCE Protected Mode<br>Driver for the Secure Crypto Engine (SCE9) on RA MCUs. |
| | Secure Crypto Engine (r_sce_protected_cavp)<br>Driver for the CAVP Certified Secure Crypto Engine (SCE) on RA MCUs. |
| | Secure Key Injection (r_rsip_key_injection)<br>Driver for the Secure Key Injection on RA MCUs. |
| | Secure Key Injection (r_sce_key_injection)<br>Driver for the Secure Key Injection on RA MCUs. |
| | TinyCrypt H/W Acceleration (rm_tinycrypt_port)<br>AES128 Hardware acceleration for TinyCrypt on the RA2 family. |

#### 5.2.15.1 AWS Device Provisioning

Modules » Security

AWS Device Provisioning example software.

# Overview

### Terminology

The terminology defined below will be used in the following sections.

| Term | Description |
|---|---|
| Service Provider | Entity that provides the cloud infrastructure and associated services, for example, AWS/Azure. |
| Device Manufacturer | Entity that provides the MCU, for example, |

| | |
|---|---|
| | Renesas. |
| OEM | Entity that uses the MCU to create a product. |
| Customer | End user of OEM product. |

**Device ID**

For systems that intend to use Public Key Certificate (PKC), the Device ID is in the form of a key pair (RSA or ECC). A PKC comprises of a **public key**, metadata, and finally a signature over all that. This signature is generated by the entity that issues the certificate and is known as a CA (Certificate Authority). The most common format for a public certificate is the X.509 format which is typically PEM (base 64) encoded such that the certificate is human-readable. It can also be DER encoded which is binary encoding and thus not human readable. The **public key** portion of the Device ID is used for the Device Certificate.

**Provisioning**

Device Provisioning refers to the process by which a service provider links a certificate to a Device ID and thus a device. Depending on the provisioning model, an existing certificate from the device may be used or a new one will be issued at this stage.  Provisioning (also referred to as Registration) occurs with respect to a particular service provider, for example, AWS or Azure. It is necessary that the certificate is issued by the service provider or a CA known to those providers. When a device is provisioned with AWS for example, the AWS IoT service associates the Device ID (and thus the device) with a specific certificate. The certificate will be programmed into the device and for all future transactions with AWS, the certificate will be used as the means of identifying the device. The public and private key are also stored on the MCU.

**Provisioning Models**

Provisioning services vary between service providers. There are essentially three general provisioning models.

1. Provisioning happens on the production line. This requires the provisioning Infrastructure to be present on the production line. This is the most secure model, but is expensive.
2. Devices are programmed with a shared credential that is linked into the code at build time and the provisioning occurs when a customer uses the device for the first time. The shared credential and a unique device serial number are used to uniquely identify the device during the provisioning process. So long as the product only has the shared credential, it will only operate with limited (as defined by certificate policy) functionality .Once the provisioning is done, then the device will be fully functional. This is the most common use case for consumer products where no sensitive information is being transmitted. AWS provides an example of this model.
3. Devices have no identity programmed in the factory; provisioning occurs through some other device like a smartphone which is already trusted by the service provider.

In all these cases, the Device Identity

1. Is unique to the device
2. Must have restricted access within the device
3. Can be used to issue more than one certificate and the certificates themselves have to be updatable in the field.

AWS uses the PKCS11 API to erase, store and retrieve certificates. These PKCS11 functions (Write, Read and Erase) are separated out into a Physical Abstraction Layer (PAL) which the OEM/Device Manufacturer is expected to implement for the type of memory that they intend to use. The internal

rm_aws_pkcs11_pal_littlefs module implements these requirements on RA MCU data flash.

# AWS Provisioning Example

AWS provides an **example** implementation to support device provisioning. This implementation uses the PKCS11 API to store device credentials into the PKCS11 defined memory. The implementation (aws_dev_mode_key_provisioning.c) exposes:

1. vAlternateKeyProvisioning()

This function requires that the device credentials be provided in PEM format. Using this example function as is in production is not recommended.

Note **vDevModeKeyProvisioning()** is no longer supported.

Credentials can be created as follows: *Create your own CA and use that to generate the device certificate. This CA will have to be registered with the service provider with which the product will be used, for example Register your CA with AWS.

- Use AWS to generate the device certificate.

# Examples

### Basic Example

This is a basic example of provisioning a device using the AWS demo implementation.

```
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
```

```
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
void device_provisioning_example (void)
{
 /* Initialize the crypto hardware acceleration. */
```

```
 mbedtls_platform_setup(NULL);
    ProvisioningParams_t params;
 /* Provision device with provided credentials. The provided credentials are written
to data flash.
  * In production, the credentials can be provided over a comms channel instead of
being linked into the image.
  * The same example provisioning function, vAlternateKeyProvisioning, can be used in
that case. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
}
```

# Limitations

The provisioning code is an example provided by AWS. It must be modified to meet product requirements.

### 5.2.15.2 Azure RTOS NetX Crypto HW Acceleration (rm_netx_secure_crypto)

Modules » Security

#### Detailed Description

Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API.

# Overview

Please refer to the NetXDuo - NetX Crypto documentation for further details.

### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E51A | RA8M1, RA8D1, RA8T1 |
| RSIP-E50D | RA8P1 |
| RSIP-E31A | RA4C1 |
| RSIP-E11A | RA4L1 |
| SCE9 | RA6M4, RA4M3, RA4M2, RA6M5 |
| SCE7 | RA6M3, RA6M2, RA6M1, RA6T1 |
| SCE5 | RA4W1, RA4M1 |
| SCE5B | RA6T2 |
| AES Engine | RA2A1, RA2E1, RA2E2, RA2L1 |
| AES Engine 2 | RA2A2 |
| TRNG | RA0E1, RA0E2, RA2L2, RA4E1, RA4E2, RA4T1, RA6E1, RA6E2, RA6T3, RA8E1, RA8E2 |

### Features

| | | AES Engine | AES Engine 2 | SCE5 | SCE5_B | SCE7 | SCE9 | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TRNG | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Hash | SHA224, SHA256 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | SHA384, SHA512 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| | SHA512/224, SHA512/256 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| MAC | HMAC-SHA224, HMAC-SHA256 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | HMAC-SHA384, HMAC-SHA512 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| | HMAC-SHA51 | – | – | – | – | – | – | – | Yes | Yes | Yes |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2/224 | | | | | | | | | |
| | HMAC-SHA512/256 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| AES | AES128-CBC, CTR | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES128-GCM (1) | – | Yes | Yes | Yes | Yes | (2) | Yes | Yes | Yes | Yes |
| | AES128-CCM (1) | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES192-CBC, CTR | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES192-GCM (1) | – | – | – | – | Yes | (2) | – | – | Yes | Yes |
| | AES192-CCM (1) | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES256-CBC, CTR | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES256-GCM (1) | – | Yes | Yes | Yes | Yes | (2) | Yes | Yes | Yes | Yes |
| | AES256-CCM (1) | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| ECC | secp256r1 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | secp384r1 | – | – | – | – | Yes | Yes | – | Yes | Yes | Yes |
| RSA | RSA-1024 | – | – | – | – | – | Yes | – | – | Yes | Yes |
| | RSA-2048 | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | RSA-3072 (3) | – | – | – | – | – | Yes | – | – | Yes | Yes |
| | RSA-4096 (3) | – | – | – | – | – | Yes | – | – | Yes | Yes |

*Note*

*1. Only block cipher unit is HW accelerated*
*2. Encrypt - Fully accelerated; Decrypt - Only GHASH and block cipher unit is HW accelerated*
*3. Signature verification only*

# Configuration

### Build Time Configurations for rm_netx_secure_crypto_sw_port

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Standalone Usage | • Use Standalone Crypto Only<br>• Use with TLS | Use Standalone Crypto Only | Defines NX_CRYPTO_STANDALONE_ENABLE. |
| Maximum RSA Modulus size (bits) | • 1024<br>• 2048<br>• 3072<br>• 4096 | 4096 | |

### Build Time Configurations for rm_netx_secure_crypto

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Hardware Acceleration | | | |
| Hardware Acceleration > Hash | | | |
| SHA256/224 | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT. |
| SHA512/384 | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA512_ALT. |
| Hardware Acceleration > Cipher | | | |
| AES | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT |
| Hardware Acceleration > Public Key Cryptography (PKC) | | | |
| Hardware Acceleration > Public Key Cryptography (PKC) > ECC | | | |
| ECC | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_M |

| | | | |
|---|---|---|---|
| | | | ETHODS_ECC_ALT |
| ECDSA Scratch Buffer Size (Bytes) | Value must be a non-negative integer | 3016 | Sets value of NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE |
| ECDH Scratch Buffer Size (Bytes) | Value must be a non-negative integer | 2464 | Sets value of NX_CRYPTO_ECDH_SCRATCH_BUFFER_SIZE |
| Hardware Acceleration > Public Key Cryptography (PKC) > RSA | | | |
| RSA | MCU Specific Options | | Enables/Disables RSA HW support |
| RSA 1024 (HW) | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_1024_ALT to allow HW support |
| RSA 2048 (HW) | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT to allow HW support |
| RSA 3072 Verify/Encryption (HW) | MCU Specific Options | | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT to allow HW support |
| RSA 4096 Verify/Encryption (HW) | MCU Specific Options | | Enables RSA NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT to allow HW support |
| RSA Scratch Buffer Size (Bytes) | MCU Specific Options | | Sets value of NX_CRYPTO_RSA_SCRATCH_BUFFER_SIZE |
| TRNG | • Enabled<br>• Disabled | Enabled | Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT. |
| Standalone Usage | • Use Standalone Crypto Only<br>• Use with TLS | Use Standalone Crypto Only | Defines NX_CRYPTO_STANDALONE_ENABLE. |

**Random Number Generator Configuration**

To enable hardware acceleration for the TRNG, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT must be defined in the configuration file. By default TRNG is enabled which can be disabled using the RA Configuration editor.

Once enabled 'rand' function will be mapped to HW TRNG; the 'srand' function is not supported, any calls to this function will have no effect. Functionality to re-seed the HW TRNG is not supported by the existing implementation.

If disabled, both 'rand' and 'srand' will be mapped to the C Standard Library. This would require

setting up the heap as 'rand' implementation calls 'malloc'.

### SHA Configuration

To enable hardware acceleration for the SHA256/224 or SHA384/512 calculation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT or NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA512_ALT must be defined in the configuration file. By default SHA is enabled which can be disabled using the RA Configuration editor.

### AES Configuration

To enable hardware acceleration for the AES128/192/256 operation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT must be defined in the configuration file. By default AES is enabled which can be disabled using the RA Configuration editor.

### ECC Configuration

To enable hardware acceleration for the ECDSA and ECDH for curves SECP384R1 and SECP256R1, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_ECC_ALT must be defined in the configuration file. By default ECC operations are enabled which can be disabled using the RA Configuration editor.

### RSA Configuration

To enable hardware acceleration for the RSA Encrypt/Decrypt (or Sign/Verify) operation(s), the macro(s) below must be must be defined in the configuration file:

| Configuration Macro | Feature / Operation |
|---|---|
| NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_1024_ALT | Signature Generation / Signature Verification (Encryption / Decryption) |
| NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT | Signature Generation / Signature Verification (Encryption / Decryption) |
| NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT | Signature Verification (Encryption Only) |
| NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT | Signature Verification (Encryption Only) |

By default RSA 2048 is enabled which can be disabled using the RA Configuration editor.

RSA software implementation is completely disabled when any of the above macros are enabled.

# Usage Notes

### Memory Alignment

Use 32bit aligned buffer pointers as arguments to APIs for best performance.

### Hardware Initialization

_nx_crypto_initialize() must be invoked before using the NetX Crypto APIs to ensure that the SCE peripheral is initialized.

### Memory Usage

Sufficient memory must be allocated to be used as 'crypto_metadata' for the chosen crypto operation(s). Refer Azure RTOS NetX Crypto API description for recommended 'crypto_metadata_size' based on selected crypto operations. Sufficient amount of memory must be allocated for the thread stack to support low level crypto operations when using this module in the standalone mode or through NetX Secure (TLS). A minimum stack of 0x1000 is required to use ECC and RSA. This is either the main stack in a bare metal application or the specific thread stack for an RTOS based application.

### AES Usage

#### GCM mode

The first byte of the IV must indicate the length of the subsequent IV. For example if the IV is {0x00, 0x00, 0x00}, then the IV pointer passed to the _nx_crypto_method_aes_operation must store the IV as {0x03, 0x00, 0x00, 0x00}. Refer to the example code for actual usage.

#### CTR mode

For CTR mode the IV pointer must be as defined in Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP) under 'Figure 2. Counter Block Format'. The IV must be 8 bytes in length. The Nonce field in the reference above must be 4 bytes and should be passed to _nx_crypto_method_aes_operation through the key pointer stored after the actual AES key. For Example, if the AES 128-bit Plain Key is {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} and the Nonce is {0x01, 0x07, 0xBD, 0xFD}, the key passed to the _nx_crypto_method_aes_operation during Encryption/Decryption should be set as {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0xBD, 0xFD}. This format would also be valid for Wrapped keys where the Nonce is appended at the end of the actual Wrapped key. Refer to the example code for actual usage.

The 'Block Counter' field in the above reference is fixed to {0x00, 0x00, 0x00, 0x01} at the beginning and increments internally after every subsequent AES block is processed. Test vectors that have the initial Block Counter not set to {0x00, 0x00, 0x00, 0x01} cannot be used in this implementation.

### ECC Usage

ECC operations include ECDH and ECDSA. As a part of ECDSA operation the input message can be hashed before signing or verification, or the message digest can be provided directly. ECC Scratch buffer size can be optionally reduced as supported ECC computations are now done by the HW. This is controlled by NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE and NX_CRYPTO_ECDH_SCRATCH_BUFFER_SIZE macros for ECDSA and ECDH respectively.

| Operation | Key Format |
|---|---|
| ECDSA Signature | Plain private key; Wrapped private key |
| ECDSA Signature-Verify | Uncompressed public key |
| Key Generation using ECDSA operation API | Wrapped private key; Uncompressed public key |
| ECDH private key import | Plain private key; Wrapped private key (Allows for Uncompressed and Formatted public key) |
| ECDH public key export | Uncompressed public key |

| | |
|---|---|
| ECDH setup | Uncompressed public key |
| ECDH shared secret calculate | Uncompressed public key |
| Key Generation using ECDH operation API | Wrapped private key; Uncompressed public key |

Note:

- Uncompressed public key is of the form (0x04 || Qx || Qy). Refer Section 2.2. Subject Public Key under RFC5480
- Formatted public key is of the form (Key Info (4 bytes) || Qx || Qy || Key Info (16 bytes)). This is the key which is used internally by SCE peripheral.

### RSA Usage

### Wrapped Key Usage

To use the NetX Crypto stack with wrapped private keys (for signature generation/decryption), a dummy pointer (non-NULL) should be passed to the 'key' parameter during _nx_crypto_method_rsa_init API call. However, the 'key_size_in_bits' parameter should be equal to the intended RSA modulus length in bits. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_rsa_operation API call with its length in bits passed through the 'key_size_in_bits' parameter.

For PKCS1V15 operation 'key' and 'key_size_in_bits' parameters of _nx_crypto_method_pkcs1_v1_5_init are unused. These can be passed as NULL and 0 respectively. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_pkcs1_v1_5_operation API call and the intended modulus length in bits must be passed through the 'key_size_in_bits' parameter.

### Software Implementation

The plaintext data passed in to the RSA encryption API must be 4-byte aligned. If it is not aligned then when the resultant encrypted data is decrypted using the decryption API, the unaligned data will be endian-swapped. For instance, if {0x01, 0x02,0x03, 0x04, 0x05} is passed as input data for encryption, when decrypted, the result will be {0x01, 0x02, 0x03, 0x04, 0x00, 0x00, 0x00, 0x05}. Note that this alignment of input data need not be taken care while using Hardware accelerated RSA.

### Limitations

- Only little endian mode is supported.
- RSA CRT keys are not supported.
- ECJPAKE related operations are unsupported for NIST 224, 256 and 384 bit curves when HW ECC is eanbled.

# Examples

### Initialization Example

This example shows how to initialize the HW crypto engine. This step must be performed before any crypto algorithm is used.

```
/* Setup the platform; initialize the SCE and the TRNG */

   err = nx_crypto_initialize();
```

```
    assert(NX_CRYPTO_SUCCESS == err);
```

### Hash Example

This is an example on calculating the SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_sha256;
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void netx_secure_crypto_sha256_example (void)
{
 size_t   actual_hash_len = RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES;
    uint8_t actual_hash[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES];
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto SHA256 init */
    err = _nx_crypto_method_sha256_init(&crypto_method_sha256, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
```

```
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
   err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                            handler,
                                            &crypto_method_sha256,
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            0,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL,
                                            0,
                                            metadata,
                                            metadata_size,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_UPDATE,
 * call this multiple times if needed to hash multiple data batches */
   err =
      _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                         handler,
                                         &crypto_method_sha256,
                                         NX_CRYPTO_NULL,
                                         0,
                                         (uint8_t *) NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200),
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL,
                                         0,
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
```

```
 /* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
    err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                             handler,
                                             &crypto_method_sha256,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             (uint8_t *) actual_hash,
                                             actual_hash_len,
                                             metadata,
                                             metadata_size,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Ensure generated SHA256 hash matches the expected digest */
    err = (uint32_t) memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len);
    assert(0 == err);
}
```

### AES Example

This is an example on using the NetX Crypto API to encrypt and decrypt multi-block data.

### AES CBC Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_cbc_256;
/* fe8901fecd3ccd2ec5fdc7c7a0b50519c245b42d611a5ef9e90268d59f3edf33 */
const uint8_t NIST_AES256_CBC_key[] =
{
    0xfe, 0x89, 0x01, 0xfe, 0xcd, 0x3c, 0xcd, 0x2e, 0xc5, 0xfd, 0xc7, 0xc7, 0xa0,
0xb5, 0x05, 0x19,
    0xc2, 0x45, 0xb4, 0x2d, 0x61, 0x1a, 0x5e, 0xf9, 0xe9, 0x02, 0x68, 0xd5, 0x9f,
0x3e, 0xdf, 0x33
```

```
};
/* 851e8764776e6796aab722dbb644ace8 */
const uint8_t NIST_AES256_CBC_iv[] =
{
    0xbd, 0x41, 0x6c, 0xb3, 0xb9, 0x89, 0x22, 0x28, 0xd8, 0xf1, 0xdf, 0x57, 0x56,
0x92, 0xe4, 0xd0
};
/* 6282b8c05c5c1530b97d4816ca434762 */
const uint8_t NIST_AES256_CBC_plaintext[] =
{
    0x8d, 0x3a, 0xa1, 0x96, 0xec, 0x3d, 0x7c, 0x9b, 0x5b, 0xb1, 0x22, 0xe7, 0xfe,
0x77, 0xfb, 0x12,
    0x95, 0xa6, 0xda, 0x75, 0xab, 0xe5, 0xd3, 0xa5, 0x10, 0x19, 0x4d, 0x3a, 0x8a,
0x41, 0x57, 0xd5,
    0xc8, 0x9d, 0x40, 0x61, 0x97, 0x16, 0x61, 0x98, 0x59, 0xda, 0x3e, 0xc9, 0xb2,
0x47, 0xce, 0xd9
};
/* 6acc04142e100a65f51b97adf5172c41 */
const uint8_t NIST_AES256_CBC_ciphertext[] =
{
    0x60, 0x8e, 0x82, 0xc7, 0xab, 0x04, 0x00, 0x7a, 0xdb, 0x22, 0xe3, 0x89, 0xa4,
0x47, 0x97, 0xfe,
    0xd7, 0xde, 0x09, 0x0c, 0x8c, 0x03, 0xca, 0x8a, 0x2c, 0x5a, 0xcd, 0x9e, 0x84,
0xdf, 0x37, 0xfb,
    0xc5, 0x8c, 0xe8, 0xed, 0xb2, 0x93, 0xe9, 0x8f, 0x02, 0xb6, 0x40, 0xd6, 0xd1,
0xd7, 0x24, 0x64
};
void netx_secure_crypto_aes256cbc_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
```

```
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_cbc_256, (uint8_t *)
NIST_AES256_CBC_key,
 sizeof(NIST_AES256_CBC_key) << 3U, &handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_cbc_256,
                                          NULL,
                                          0,
                                          (uint8_t *) NIST_AES256_CBC_plaintext,
 sizeof(NIST_AES256_CBC_plaintext),
                                          (uint8_t *) NIST_AES256_CBC_iv,
                                          generated_ciphertext,
 sizeof(generated_ciphertext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, NIST_AES256_CBC_ciphertext,
sizeof(generated_ciphertext));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_aes_cbc_256,
```

```
                                        NULL,
                                        0,
                                        (uint8_t *) NIST_AES256_CBC_ciphertext,
 sizeof(NIST_AES256_CBC_ciphertext),
                                        (uint8_t *) NIST_AES256_CBC_iv,
                                        generated_plaintext,
 sizeof(generated_plaintext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, NIST_AES256_CBC_plaintext,
sizeof(generated_ciphertext));
    assert(0 == err);
}
```

### AES GCM Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_128_gcm_16;
/* 83F9D97D4AB759FDDCC3EF54A0E2A8EC */
static const uint8_t key_gcm_128[] =
{
    0x83, 0xF9, 0xD9, 0x7D, 0x4A, 0xB7, 0x59, 0xFD, 0xDC, 0xC3, 0xEF, 0x54, 0xA0,
0xE2, 0xA8, 0xEC
};
/* In case of IV the IV length must be the first byte followed by the actual IV.
 * In this example the IV length is 0x01 and the actual IV is 0xCF
 */
/* 01CF */
static const uint8_t iv_gcm_128[] =
{
    0x01, 0xCF
};
```

```
/* 77E6329CF9424F71C808DF9170BFD298 */
static const uint8_t plain_gcm_128[] =
{
    0x77, 0xE6, 0x32, 0x9C, 0xF9, 0x42, 0x4F, 0x71, 0xC8, 0x08, 0xDF, 0x91, 0x70,
0xBF, 0xD2, 0x98
};
/* 6DD49EAEB4103DAC8F97E3234946DD2D */
static const uint8_t aad_gcm_128[] =
{
    0x6D, 0xD4, 0x9E, 0xAE, 0xB4, 0x10, 0x3D, 0xAC, 0x8F, 0x97, 0xE3, 0x23, 0x49,
0x46, 0xDD, 0x2D
};
/* 50DE86A7A92A8A5EA33DB5696B96CD77AA181E84BC8B4BF5A68927C409D422CB */
static const uint8_t secret_gcm_128[] =
{
 /* Ciphertext */
    0x50, 0xDE, 0x86, 0xA7, 0xA9, 0x2A, 0x8A, 0x5E, 0xA3, 0x3D, 0xB5, 0x69, 0x6B,
0x96, 0xCD, 0x77,
 /* Tag */
    0xAA, 0x18, 0x1E, 0x84, 0xBC, 0x8B, 0x4B, 0xF5, 0xA6, 0x89, 0x27, 0xC4, 0x09,
0xD4, 0x22, 0xCB
};
void netx_secure_crypto_aes128gcm_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
```

```
   err =
      _nx_crypto_method_aes_init(&crypto_method_aes_128_gcm_16,
                                 (uint8_t *) key_gcm_128,
sizeof(key_gcm_128) << 3U,
           &handler,
           metadata,
           metadata_size);
   assert(NX_CRYPTO_SUCCESS == err);
/* Setup Additional Authentication Data */
   err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                         handler,
                                         &crypto_method_aes_128_gcm_16,
                                         NULL,
                                         0,
                                         (uint8_t *) aad_gcm_128,
sizeof(aad_gcm_128),
                                         NULL,
                                         NULL,
                                         0,
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Encryption. */
   err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                         handler,
                                         &crypto_method_aes_128_gcm_16,
                                         NULL,
                                         0,
                                         (uint8_t *) plain_gcm_128,
sizeof(plain_gcm_128),
                                         (uint8_t *) iv_gcm_128,
                                         generated_ciphertext,
```

```
 sizeof(generated_ciphertext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* The 16 byte tag is appended to the generated ciphertext */
 /* Verify generated tag matches the expected tag */
    err = (uint32_t) memcmp(&generated_ciphertext[sizeof(plain_gcm_128)],
&secret_gcm_128[sizeof(plain_gcm_128)], 16U);
    assert(0 == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_gcm_128, sizeof
(secret_gcm_128));
    assert(0 == err);
 /* Setup Additional Authentication Data */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                        handler,
                                        &crypto_method_aes_128_gcm_16,
                                        NULL,
                                        0,
                                        (uint8_t *) aad_gcm_128,
 sizeof(aad_gcm_128),
                                        NULL,
                                        NULL,
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                        handler,
```

```
                                        &crypto_method_aes_128_gcm_16,
                                        NULL,
                                        0,
                                        (uint8_t *) generated_ciphertext,
 sizeof(secret_gcm_128), /* ciphertext size + tag size */
                                        (uint8_t *) iv_gcm_128,
                                        generated_plaintext,
 sizeof(generated_plaintext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, plain_gcm_128, sizeof
(plain_gcm_128));
    assert(0 == err);
}
```

### AES CTR Example

```
NX_CRYPTO_METHOD crypto_method_aes_ctr_256 =
{
    NX_CRYPTO_ENCRYPTION_AES_CTR,           /* AES crypto
algorithm                   */
    NX_CRYPTO_AES_256_KEY_LEN_IN_BITS,      /* Key size in bits
*/
    NX_CRYPTO_AES_IV_LEN_IN_BITS,           /* IV size in
bits                        */
    0,                                      /* ICV size in bits, not
used             */
    (NX_CRYPTO_AES_BLOCK_SIZE_IN_BITS >> 3), /* Block size in bytes
*/
 sizeof(NX_CRYPTO_AES),                    /* Metadata size in bytes                */
    _nx_crypto_method_aes_init,             /* AES-CBC initialization
```

```
routine         */
    _nx_crypto_method_aes_cleanup,          /* AES-CBC cleanup routine
*/
    _nx_crypto_method_aes_ctr_operation      /* AES-CBC
operation                       */
};
/*Note: For CTR, the key_ctr is the conjunction of key and nonce. */
/* D0E78C4D0B30D33F5BF4A132B2F94A4A38963511A3904B117E35A37B5AAC8A193BF0D158 */
const uint8_t key_ctr_256[] =
{
 /* AES Key */
    0xD0, 0xE7, 0x8C, 0x4D, 0x0B, 0x30, 0xD3, 0x3F, 0x5B, 0xF4, 0xA1, 0x32, 0xB2,
0xF9, 0x4A, 0x4A,
    0x38, 0x96, 0x35, 0x11, 0xA3, 0x90, 0x4B, 0x11, 0x7E, 0x35, 0xA3, 0x7B, 0x5A,
0xAC, 0x8A, 0x19,
 /* Nonce */
    0x3B, 0xF0, 0xD1, 0x58,
};
/* A1A31704C8B7E16C */
const uint8_t iv_ctr_256[] =
{
    0xA1, 0xA3, 0x17, 0x04, 0xC8, 0xB7, 0xE1, 0x6C,
};
/* 981FA33222C5451017530155A4BF7F29 */
const uint8_t plain_ctr_256[] =
{
    0x98, 0x1F, 0xA3, 0x32, 0x22, 0xC5, 0x45, 0x10, 0x17, 0x53, 0x01, 0x55, 0xA4,
0xBF, 0x7F, 0x29,
};
/* 643B91B4E541B20AAAEAB77F2D328566 */
const uint8_t secret_ctr_256[] =
{
    0x64, 0x3B, 0x91, 0xB4, 0xE5, 0x41, 0xB2, 0x0A, 0xAA, 0xEA, 0xB7, 0x7F, 0x2D,
0x32, 0x85, 0x66,
```

```
};
void netx_secure_crypto_aes256ctr_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_ctr_256,
                                  (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
             &handler,
             metadata,
             metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_ctr_256,
                                    (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                          (uint8_t *) plain_ctr_256,
 sizeof(plain_ctr_256),
                                          (uint8_t *) iv_ctr_256,
                                          generated_ciphertext,
 sizeof(generated_ciphertext),
```

```
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_ctr_256, sizeof
(secret_ctr_256));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                        handler,
                                        &crypto_method_aes_ctr_256,
                                    (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                        (uint8_t *) secret_ctr_256,
 sizeof(secret_ctr_256),
                                        (uint8_t *) iv_ctr_256,
                                        generated_plaintext,
 sizeof(generated_plaintext),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, plain_ctr_256, sizeof
(plain_ctr_256));
    assert(0 == err);
}
```

### ECDSA Example

This is an example on using the NetX Crypto API to sign and verify input message data. Based on the

hash algorithm selected a digest is computed of the plain input message before sign/verify.

```
extern NX_CRYPTO_METHOD crypto_method_ecdsa;
extern NX_CRYPTO_METHOD crypto_method_ec_secp256;
extern NX_CRYPTO_METHOD crypto_method_sha256;
const uint8_t ECC_SECP256R1Keydata[] =
{
    0xf9, 0xa7, 0x68, 0x71, 0x24, 0x68, 0x9d, 0x32, 0x92, 0x6f, 0x1d, 0xfb, 0xbe,
0xf2, 0x61, 0x41, // NOLINT(readability-magic-numbers)
    0x07, 0x54, 0x0d, 0xb9, 0xa8, 0x8a, 0x8b, 0xc2, 0xd5, 0xe9, 0x38, 0x4b, 0xf9,
0xe5, 0x43, 0x5a // NOLINT(readability-magic-numbers)
};
const uint8_t ECC_SECP256R1PublicKeydata[] =
{
    0x04,
                /* ASN1 Constant */
    0x5b, 0xba, 0xd4, 0x2e, 0xb5, 0xc1, 0x07, 0xf2, 0x0e, 0x01, 0x95, 0x42, 0x6e,
0x90, 0xb8, 0x4e, // NOLINT(readability-magic-numbers)
    0xe9, 0x5a, 0xa1, 0xe8, 0x4c, 0x6c, 0xa5, 0x32, 0x3c, 0xf3, 0x09, 0xf5, 0xff,
0x8b, 0x3d, 0x26, // NOLINT(readability-magic-numbers)
    0xb6, 0x88, 0xc1, 0xdb, 0x02, 0xaf, 0x4d, 0xa5, 0x0e, 0x73, 0x61, 0x96, 0xb3,
0x59, 0x95, 0x6f, // NOLINT(readability-magic-numbers)
    0x5e, 0xc9, 0xa1, 0xf9, 0xb7, 0xb3, 0xb6, 0xdf, 0x54, 0x82, 0x79, 0xe3, 0xb6,
0x4e, 0xac, 0xb6 // NOLINT(readability-magic-numbers)
};
const uint8_t ECC_SECP256R1Message[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
void netx_secure_crypto_ecdsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDSA);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
    ULONG sig_length;
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
```

```
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdsa_init(&crypto_method_ecdsa, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set hash method. */
    err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_HASH_METHOD_SET,
                                            handler,
                                           &crypto_method_ecdsa,
                                            NX_CRYPTO_NULL,
                                            0,
                                           (uint8_t *) &crypto_method_sha256,
 sizeof(NX_CRYPTO_METHOD *),
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL,
                                            0,
                                            metadata,
                                            metadata_size,
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err =
       _nx_crypto_method_ecdsa_operation(NX_CRYPTO_EC_CURVE_SET,
                                         handler,
                                         &crypto_method_ecdsa,
                                         NX_CRYPTO_NULL,
                                         0,
                                         (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL,
```

```
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   extended_output.nx_crypto_extended_output_data          = output;
   extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
/* Sign the hash data using ECDSA. */
   err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_GENERATE,
                                           handler,
                                          &crypto_method_ec_secp256,
                                          (uint8_t *) ECC_SECP256R1Keydata,
sizeof(ECC_SECP256R1Keydata) << 3,
                                          (uint8_t *) ECC_SECP256R1Message,
sizeof(ECC_SECP256R1Message),
      NX_CRYPTO_NULL,
                                          (uint8_t *) &extended_output,
sizeof(extended_output),
      metadata,
      metadata_size,
      NX_CRYPTO_NULL,
      NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   sig_length = extended_output.nx_crypto_extended_output_actual_size;
/* Verify the generated signature. */
   err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_VERIFY,
                                           handler,
                                          &crypto_method_ec_secp256,
                                          (uint8_t *) ECC_SECP256R1PublicKeydata,
sizeof(ECC_SECP256R1PublicKeydata) << 3,
                                          (uint8_t *) ECC_SECP256R1Message,
sizeof(ECC_SECP256R1Message),
      NX_CRYPTO_NULL,
```

```
        output,
        sig_length,
        metadata,
        metadata_size,
        NX_CRYPTO_NULL,
        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
}
```

### ECDH Example

This is an example on using the NetX Crypto API to generate a shared secret using ECDH. A shared secret is computed using known public key (from peer) and generated private key. Another shared secret is computed using the generated public key and known private key (imported to mimic peer). Both the shared secrets are checked to be the same.

```
extern NX_CRYPTO_METHOD crypto_method_ecdh;
/*Private key 59137e38152350b195c9718d39673d519838055ad908dd4757152fd8255c09bf */
const uint8_t ECC_SECP256R1Keydata_ecdh[] =
{
    0x59, 0x13, 0x7e, 0x38, 0x15, 0x23, 0x50, 0xb1, 0x95, 0xc9, 0x71, 0x8d, 0x39,
0x67, 0x3d, 0x51, // NOLINT(readability-magic-numbers)
    0x98, 0x38, 0x05, 0x5a, 0xd9, 0x08, 0xdd, 0x47, 0x57, 0x15, 0x2f, 0xd8, 0x25,
0x5c, 0x09, 0xbf, // NOLINT(readability-magic-numbers)
};
/*Public key 4, a8c5fdce8b62c5ada598f141adb3b26cf254c280b2857a63d2ad783a73115f6b,
806e1aafec4af80a0d786b3de45375b517a7e5b51ffb2c356537c9e6ef227d4a*/
const uint8_t ECC_SECP256R1PublicKeydata_ecdh[] =
{
    0x04,
    0xa8,0xc5, 0xfd, 0xce, 0x8b, 0x62, 0xc5, 0xad, 0xa5, 0x98, 0xf1, 0x41, 0xad,
0xb3, 0xb2, 0x6c, // NOLINT(readability-magic-numbers)
    0xf2,0x54, 0xc2, 0x80, 0xb2, 0x85, 0x7a, 0x63, 0xd2, 0xad, 0x78, 0x3a, 0x73,
0x11, 0x5f, 0x6b, // NOLINT(readability-magic-numbers)
    0x80,0x6e, 0x1a, 0xaf, 0xec, 0x4a, 0xf8, 0x0a, 0x0d, 0x78, 0x6b, 0x3d, 0xe4,
0x53, 0x75, 0xb5, // NOLINT(readability-magic-numbers)
```

```
    0x17,0xa7, 0xe5, 0xb5, 0x1f, 0xfb, 0x2c, 0x35, 0x65, 0x37, 0xc9, 0xe6, 0xef,
0x22, 0x7d, 0x4a, // NOLINT(readability-magic-numbers)
};
void netx_secure_crypto_ecdh_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDH)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDH);
    uint32_t err          = NX_CRYPTO_SUCCESS;
    uint8_t local_public_key[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    uint32_t local_public_key_len = 0;
    uint8_t shared_secret[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    uint32_t shared_secret_len = 0;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
```

```
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Generate local public key. This will generate a key pair.
  * The private wrapped key will be held by the ecdh context and the public key
(local_public_key)
  * will be returned for sharing with the peer.
  */
    extended_output.nx_crypto_extended_output_data          = local_public_key;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof
(local_public_key);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_SETUP,
                                        NX_CRYPTO_NULL,
                                        &crypto_method_ecdh,
                                        NX_CRYPTO_NULL,
                                        0,
                                        NX_CRYPTO_NULL,
                                        0,
                                        NX_CRYPTO_NULL,
                                        (uint8_t *) &extended_output,
 sizeof(extended_output),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    local_public_key_len = extended_output.nx_crypto_extended_output_actual_size;
 /* Calculate shared secret using the test (peer's) public key. */
    extended_output.nx_crypto_extended_output_data          = shared_secret;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(shared_secret);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                        NX_CRYPTO_NULL,
                                        &crypto_method_ecdh,
                                        NX_CRYPTO_NULL,
```

```
                                        0,
                                        (uint8_t *)
ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
                                        NX_CRYPTO_NULL,
                                        (uint8_t *) &extended_output,
 sizeof(extended_output),
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    shared_secret_len = extended_output.nx_crypto_extended_output_actual_size;
    err = _nx_crypto_method_ecdh_cleanup(metadata);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify. The below operations will be carried out by the peer. */
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                        NX_CRYPTO_NULL,
                                        &crypto_method_ecdh,
                                        NX_CRYPTO_NULL,
                                        0,
                                        (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL,
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
```

```
                                        NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Import the test private key. The peer could generate its own key pair,
  * in this example a test private key is used for simplicity. */
    err =
       _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_KEY_PAIR_IMPORT,
         NX_CRYPTO_NULL,
                                       &crypto_method_ecdh,
                                       (uint8_t *) ECC_SECP256R1Keydata_ecdh,
                                       (NX_CRYPTO_KEY_SIZE)
(sizeof(ECC_SECP256R1Keydata_ecdh) << 3),
                                       (uint8_t *) ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
         NX_CRYPTO_NULL,
         NX_CRYPTO_NULL,
                                       0,
              metadata,
         metadata_size,
         NX_CRYPTO_NULL,
         NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Calculate the shared secret using the local public key generated above and shared
with the peer. */
    extended_output.nx_crypto_extended_output_data          = output;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                          NX_CRYPTO_NULL,
                                          &crypto_method_ecdh,
                                          NX_CRYPTO_NULL,
                                          0,
                                          local_public_key,
                                          local_public_key_len,
                                          NX_CRYPTO_NULL,
                                          (uint8_t *) &extended_output,
```

```
 sizeof(extended_output),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Validate the output. Both the parties must generate the same shared secret */
    err = (extended_output.nx_crypto_extended_output_actual_size !=
shared_secret_len);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, shared_secret,
extended_output.nx_crypto_extended_output_actual_size);
    assert(NX_CRYPTO_SUCCESS == err);
}
```

### RSA Example

This is an example on using the NetX Crypto API to encrypt and decrypt input message data.

```
extern NX_CRYPTO_METHOD crypto_method_rsa;
/* 00010001 */
const uint8_t public_e[] =
{
    0x00, 0x01, 0x00, 0x01,
};
/* 13FF7429F8E851F1079CCFCE3B3CD8606ABA8607AD85CBB3057501EBD58811F3C04823171F192C048E
1E883AF8CF958810151D3874AEDC8EC4F88D2065C581569F1E200852DD40B6DFD1652659085A9DD1D3B86
9EA3617D904D209DE156A60BA5929D02F16430273D10720C2F28D2B95684DCAA6B9F6A508EA2CBBC11B9F
3F30D6201EA6CFFBBF1C44255CEC58EE70DBC872442BCCF115D8F743557B5DE5F42DDDA6CEAE7977793CC
9D90ADFE65E520F5520B615CF3B8C2DC82D7AC75EDB1297CF38AB23A37EED18D4DD45D9AD051B26401BE8
6E8C8E53F9585A702D02F1B5BD65F6739DFA6BFFE560CA130B6F1D4779C556C06D9CD29FB72D8851904F9
CDEE9 */
const uint8_t private_e_2048[] =
{
    0x13, 0xFF, 0x74, 0x29, 0xF8, 0xE8, 0x51, 0xF1, 0x07, 0x9C, 0xCF, 0xCE, 0x3B,
```

```
0x3C, 0xD8, 0x60,
    0x6A, 0xBA, 0x86, 0x07, 0xAD, 0x85, 0xCB, 0xB3, 0x05, 0x75, 0x01, 0xEB, 0xD5,
0x88, 0x11, 0xF3,
    0xC0, 0x48, 0x23, 0x17, 0x1F, 0x19, 0x2C, 0x04, 0x8E, 0x1E, 0x88, 0x3A, 0xF8,
0xCF, 0x95, 0x88,
    0x10, 0x15, 0x1D, 0x38, 0x74, 0xAE, 0xDC, 0x8E, 0xC4, 0xF8, 0x8D, 0x20, 0x65,
0xC5, 0x81, 0x56,
    0x9F, 0x1E, 0x20, 0x08, 0x52, 0xDD, 0x40, 0xB6, 0xDF, 0xD1, 0x65, 0x26, 0x59,
0x08, 0x5A, 0x9D,
    0xD1, 0xD3, 0xB8, 0x69, 0xEA, 0x36, 0x17, 0xD9, 0x04, 0xD2, 0x09, 0xDE, 0x15,
0x6A, 0x60, 0xBA,
    0x59, 0x29, 0xD0, 0x2F, 0x16, 0x43, 0x02, 0x73, 0xD1, 0x07, 0x20, 0xC2, 0xF2,
0x8D, 0x2B, 0x95,
    0x68, 0x4D, 0xCA, 0xA6, 0xB9, 0xF6, 0xA5, 0x08, 0xEA, 0x2C, 0xBB, 0xC1, 0x1B,
0x9F, 0x3F, 0x30,
    0xD6, 0x20, 0x1E, 0xA6, 0xCF, 0xFB, 0xBF, 0x1C, 0x44, 0x25, 0x5C, 0xEC, 0x58,
0xEE, 0x70, 0xDB,
    0xC8, 0x72, 0x44, 0x2B, 0xCC, 0xF1, 0x15, 0xD8, 0xF7, 0x43, 0x55, 0x7B, 0x5D,
0xE5, 0xF4, 0x2D,
    0xDD, 0xA6, 0xCE, 0xAE, 0x79, 0x77, 0x79, 0x3C, 0xC9, 0xD9, 0x0A, 0xDF, 0xE6,
0x5E, 0x52, 0x0F,
    0x55, 0x20, 0xB6, 0x15, 0xCF, 0x3B, 0x8C, 0x2D, 0xC8, 0x2D, 0x7A, 0xC7, 0x5E,
0xDB, 0x12, 0x97,
    0xCF, 0x38, 0xAB, 0x23, 0xA3, 0x7E, 0xED, 0x18, 0xD4, 0xDD, 0x45, 0xD9, 0xAD,
0x05, 0x1B, 0x26,
    0x40, 0x1B, 0xE8, 0x6E, 0x8C, 0x8E, 0x53, 0xF9, 0x58, 0x5A, 0x70, 0x2D, 0x02,
0xF1, 0xB5, 0xBD,
    0x65, 0xF6, 0x73, 0x9D, 0xFA, 0x6B, 0xFF, 0xE5, 0x60, 0xCA, 0x13, 0x0B, 0x6F,
0x1D, 0x47, 0x79,
    0xC5, 0x56, 0xC0, 0x6D, 0x9C, 0xD2, 0x9F, 0xB7, 0x2D, 0x88, 0x51, 0x90, 0x4F,
0x9C, 0xDE, 0xE9,
};
/* E0F5059966A8AEC4BF7CDAC8AE2430BDF61C54D09CAB9963CBF9A52AC641E384B6431D3B6A9D181151
9A2904E1170A44446C80E7638A4AF2720A7654AB740D8A151FDD216F3D6933422FD9AC14AEDE9CCD021EA
```

```
79E46925F4B18FD1AF2C0073CFC3A69AC71A2B3673D08136CDB01C379892601C7C857D68018DAE924CB8C
D29377A14C752B92BAFF14C3A49725AE2FEFAAD4686D8A7D9F94EB11BF81E05BD5D2586526FB129E73539
F9223D496B2ACA23CCACC34D5B18533BD0F5815A76F94F4F55D965FE61599B44BD8FBAD35F42B612A4C4F
2765B2097A5C0090EA8166D9C6DA1E03B6119736B794600491C48433132D0F15D5DE3BB4270DF6BC9012B
74931 */
const uint8_t m_2048[] =
{
    0xE0, 0xF5, 0x05, 0x99, 0x66, 0xA8, 0xAE, 0xC4, 0xBF, 0x7C, 0xDA, 0xC8, 0xAE,
0x24, 0x30, 0xBD,
    0xF6, 0x1C, 0x54, 0xD0, 0x9C, 0xAB, 0x99, 0x63, 0xCB, 0xF9, 0xA5, 0x2A, 0xC6,
0x41, 0xE3, 0x84,
    0xB6, 0x43, 0x1D, 0x3B, 0x6A, 0x9D, 0x18, 0x11, 0x51, 0x9A, 0x29, 0x04, 0xE1,
0x17, 0x0A, 0x44,
    0x44, 0x6C, 0x80, 0xE7, 0x63, 0x8A, 0x4A, 0xF2, 0x72, 0x0A, 0x76, 0x54, 0xAB,
0x74, 0x0D, 0x8A,
    0x15, 0x1F, 0xDD, 0x21, 0x6F, 0x3D, 0x69, 0x33, 0x42, 0x2F, 0xD9, 0xAC, 0x14,
0xAE, 0xDE, 0x9C,
    0xCD, 0x02, 0x1E, 0xA7, 0x9E, 0x46, 0x92, 0x5F, 0x4B, 0x18, 0xFD, 0x1A, 0xF2,
0xC0, 0x07, 0x3C,
    0xFC, 0x3A, 0x69, 0xAC, 0x71, 0xA2, 0xB3, 0x67, 0x3D, 0x08, 0x13, 0x6C, 0xDB,
0x01, 0xC3, 0x79,
    0x89, 0x26, 0x01, 0xC7, 0xC8, 0x57, 0xD6, 0x80, 0x18, 0xDA, 0xE9, 0x24, 0xCB,
0x8C, 0xD2, 0x93,
    0x77, 0xA1, 0x4C, 0x75, 0x2B, 0x92, 0xBA, 0xFF, 0x14, 0xC3, 0xA4, 0x97, 0x25,
0xAE, 0x2F, 0xEF,
    0xAA, 0xD4, 0x68, 0x6D, 0x8A, 0x7D, 0x9F, 0x94, 0xEB, 0x11, 0xBF, 0x81, 0xE0,
0x5B, 0xD5, 0xD2,
    0x58, 0x65, 0x26, 0xFB, 0x12, 0x9E, 0x73, 0x53, 0x9F, 0x92, 0x23, 0xD4, 0x96,
0xB2, 0xAC, 0xA2,
    0x3C, 0xCA, 0xCC, 0x34, 0xD5, 0xB1, 0x85, 0x33, 0xBD, 0x0F, 0x58, 0x15, 0xA7,
0x6F, 0x94, 0xF4,
    0xF5, 0x5D, 0x96, 0x5F, 0xE6, 0x15, 0x99, 0xB4, 0x4B, 0xD8, 0xFB, 0xAD, 0x35,
0xF4, 0x2B, 0x61,
    0x2A, 0x4C, 0x4F, 0x27, 0x65, 0xB2, 0x09, 0x7A, 0x5C, 0x00, 0x90, 0xEA, 0x81,
```

```
0x66, 0xD9, 0xC6,
    0xDA, 0x1E, 0x03, 0xB6, 0x11, 0x97, 0x36, 0xB7, 0x94, 0x60, 0x04, 0x91, 0xC4,
0x84, 0x33, 0x13,
    0x2D, 0x0F, 0x15, 0xD5, 0xDE, 0x3B, 0xB4, 0x27, 0x0D, 0xF6, 0xBC, 0x90, 0x12,
0xB7, 0x49, 0x31,
};
/* 551C2E268F7ED44D0E8B063F5B2B510CB809F53BD54E9956971E243B2363DA123C29AB4A009EDE1FCE
C54625971A4E3490F3EA398BF7386AAC34720E43FB0C795445B520AEE4D7694EE1474F60F77E1B5F09FE2
ED004333658D212122F040322D1564512A1540400F27E18049A762A5EDC9F072CA4F49F408252D42B31BC
35523373740E90DDDA6A8CE7865EEB7C694A662C74412406AB190FE0435DA2551F0C24A48939DDA58A023
9706D40B4977473689DC36CE5A4DF4EF892816CBDE2780D9389B7384674C93B1DDAF728F292B5671679FC
7175AC0A3B2197B809E7CF410417010F3B1316D10D82466C62F3A01667B70A714E0499400E255D4C39EA7
DE55C */
const uint8_t plain_2048[] =
{
    0x55, 0x1C, 0x2E, 0x26, 0x8F, 0x7E, 0xD4, 0x4D, 0x0E, 0x8B, 0x06, 0x3F, 0x5B,
0x2B, 0x51, 0x0C,
    0xB8, 0x09, 0xF5, 0x3B, 0xD5, 0x4E, 0x99, 0x56, 0x97, 0x1E, 0x24, 0x3B, 0x23,
0x63, 0xDA, 0x12,
    0x3C, 0x29, 0xAB, 0x4A, 0x00, 0x9E, 0xDE, 0x1F, 0xCE, 0xC5, 0x46, 0x25, 0x97,
0x1A, 0x4E, 0x34,
    0x90, 0xF3, 0xEA, 0x39, 0x8B, 0xF7, 0x38, 0x6A, 0xAC, 0x34, 0x72, 0x0E, 0x43,
0xFB, 0x0C, 0x79,
    0x54, 0x45, 0xB5, 0x20, 0xAE, 0xE4, 0xD7, 0x69, 0x4E, 0xE1, 0x47, 0x4F, 0x60,
0xF7, 0x7E, 0x1B,
    0x5F, 0x09, 0xFE, 0x2E, 0xD0, 0x04, 0x33, 0x36, 0x58, 0xD2, 0x12, 0x12, 0x2F,
0x04, 0x03, 0x22,
    0xD1, 0x56, 0x45, 0x12, 0xA1, 0x54, 0x04, 0x00, 0xF2, 0x7E, 0x18, 0x04, 0x9A,
0x76, 0x2A, 0x5E,
    0xDC, 0x9F, 0x07, 0x2C, 0xA4, 0xF4, 0x9F, 0x40, 0x82, 0x52, 0xD4, 0x2B, 0x31,
0xBC, 0x35, 0x52,
    0x33, 0x73, 0x74, 0x0E, 0x90, 0xDD, 0xDA, 0x6A, 0x8C, 0xE7, 0x86, 0x5E, 0xEB,
0x7C, 0x69, 0x4A,
    0x66, 0x2C, 0x74, 0x41, 0x24, 0x06, 0xAB, 0x19, 0x0F, 0xE0, 0x43, 0x5D, 0xA2,
```

```
0x55, 0x1F, 0x0C,
    0x24, 0xA4, 0x89, 0x39, 0xDD, 0xA5, 0x8A, 0x02, 0x39, 0x70, 0x6D, 0x40, 0xB4,
0x97, 0x74, 0x73,
    0x68, 0x9D, 0xC3, 0x6C, 0xE5, 0xA4, 0xDF, 0x4E, 0xF8, 0x92, 0x81, 0x6C, 0xBD,
0xE2, 0x78, 0x0D,
    0x93, 0x89, 0xB7, 0x38, 0x46, 0x74, 0xC9, 0x3B, 0x1D, 0xDA, 0xF7, 0x28, 0xF2,
0x92, 0xB5, 0x67,
    0x16, 0x79, 0xFC, 0x71, 0x75, 0xAC, 0x0A, 0x3B, 0x21, 0x97, 0xB8, 0x09, 0xE7,
0xCF, 0x41, 0x04,
    0x17, 0x01, 0x0F, 0x3B, 0x13, 0x16, 0xD1, 0x0D, 0x82, 0x46, 0x6C, 0x62, 0xF3,
0xA0, 0x16, 0x67,
    0xB7, 0x0A, 0x71, 0x4E, 0x04, 0x99, 0x40, 0x0E, 0x25, 0x5D, 0x4C, 0x39, 0xEA,
0x7D, 0xE5, 0x5C,
};
/* 10F904E071338569EC131401A7869F42F3BCAE252B5D3C8755FD24D47997A9CD4221D992B2871E0528
3B98841FC5C379C5D0E35B3938279B344299C3CF1566E0C994D0A9013AF64174F1379A4B5E4E9DE57491F
3078F6D10011EA55535D0763E538662C9996F4FCF8B64A768685AA417ADB6978743D3D1F513CF143DD6D3
83AD6357728A88928D39E27EA4D0B2AF92FC7F63875F9D6A70FAE7993C1FF04DF9A2F99216874BC123D4B
7DA7E7E8974CFC10ACF0C7BC8747526A8D16791F969082EA9B0C36D77B67C37B325682D74178E4234D52D
5635273301A6CC35E315AE74D659B1433576DAAE6780FA39E0550D971F2CB5817CAAFC24B5220E21C8CEE
E85DD */
const uint8_t secret_2048[] =
{
    0x10, 0xF9, 0x04, 0xE0, 0x71, 0x33, 0x85, 0x69, 0xEC, 0x13, 0x14, 0x01, 0xA7,
0x86, 0x9F, 0x42,
    0xF3, 0xBC, 0xAE, 0x25, 0x2B, 0x5D, 0x3C, 0x87, 0x55, 0xFD, 0x24, 0xD4, 0x79,
0x97, 0xA9, 0xCD,
    0x42, 0x21, 0xD9, 0x92, 0xB2, 0x87, 0x1E, 0x05, 0x28, 0x3B, 0x98, 0x84, 0x1F,
0xC5, 0xC3, 0x79,
    0xC5, 0xD0, 0xE3, 0x5B, 0x39, 0x38, 0x27, 0x9B, 0x34, 0x42, 0x99, 0xC3, 0xCF,
0x15, 0x66, 0xE0,
    0xC9, 0x94, 0xD0, 0xA9, 0x01, 0x3A, 0xF6, 0x41, 0x74, 0xF1, 0x37, 0x9A, 0x4B,
0x5E, 0x4E, 0x9D,
    0xE5, 0x74, 0x91, 0xF3, 0x07, 0x8F, 0x6D, 0x10, 0x01, 0x1E, 0xA5, 0x55, 0x35,
```

```
0xD0, 0x76, 0x3E,
    0x53, 0x86, 0x62, 0xC9, 0x99, 0x6F, 0x4F, 0xCF, 0x8B, 0x64, 0xA7, 0x68, 0x68,
0x5A, 0xA4, 0x17,
    0xAD, 0xB6, 0x97, 0x87, 0x43, 0xD3, 0xD1, 0xF5, 0x13, 0xCF, 0x14, 0x3D, 0xD6,
0xD3, 0x83, 0xAD,
    0x63, 0x57, 0x72, 0x8A, 0x88, 0x92, 0x8D, 0x39, 0xE2, 0x7E, 0xA4, 0xD0, 0xB2,
0xAF, 0x92, 0xFC,
    0x7F, 0x63, 0x87, 0x5F, 0x9D, 0x6A, 0x70, 0xFA, 0xE7, 0x99, 0x3C, 0x1F, 0xF0,
0x4D, 0xF9, 0xA2,
    0xF9, 0x92, 0x16, 0x87, 0x4B, 0xC1, 0x23, 0xD4, 0xB7, 0xDA, 0x7E, 0x7E, 0x89,
0x74, 0xCF, 0xC1,
    0x0A, 0xCF, 0x0C, 0x7B, 0xC8, 0x74, 0x75, 0x26, 0xA8, 0xD1, 0x67, 0x91, 0xF9,
0x69, 0x08, 0x2E,
    0xA9, 0xB0, 0xC3, 0x6D, 0x77, 0xB6, 0x7C, 0x37, 0xB3, 0x25, 0x68, 0x2D, 0x74,
0x17, 0x8E, 0x42,
    0x34, 0xD5, 0x2D, 0x56, 0x35, 0x27, 0x33, 0x01, 0xA6, 0xCC, 0x35, 0xE3, 0x15,
0xAE, 0x74, 0xD6,
    0x59, 0xB1, 0x43, 0x35, 0x76, 0xDA, 0xAE, 0x67, 0x80, 0xFA, 0x39, 0xE0, 0x55,
0x0D, 0x97, 0x1F,
    0x2C, 0xB5, 0x81, 0x7C, 0xAA, 0xFC, 0x24, 0xB5, 0x22, 0x0E, 0x21, 0xC8, 0xCE,
0xEE, 0x85, 0xDD,
};
void netx_secure_crypto_rsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_RSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_RSA);
    uint32_t err          = NX_CRYPTO_SUCCESS;
 void   * handler       = NX_CRYPTO_NULL;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err =
```

```
      _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                 (uint8_t *) m_2048,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
             &handler,
             metadata,
             metadata_size);
   assert(NX_CRYPTO_SUCCESS == err);
   err = _nx_crypto_method_rsa_operation(NX_CRYPTO_ENCRYPT,
                                         handler,
                                         &crypto_method_rsa,
                                         (uint8_t *) public_e,


                    RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(public_e)),
                                         (uint8_t *) plain_2048,
 sizeof(m_2048),
                                         NX_CRYPTO_NULL,
        output,
 sizeof(m_2048),
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   err = (uint32_t) memcmp(output, secret_2048, sizeof(m_2048));
   assert(0 == err);
   err = _nx_crypto_method_rsa_cleanup(metadata);
   assert(NX_CRYPTO_SUCCESS == err);
 /* Decryption. */
   memset(output, 0, sizeof(output));
   err =
      _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                 (uint8_t *) m_2048,
```

```
RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
             &handler,
             metadata,
             metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
    err = _nx_crypto_method_rsa_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_rsa,
                                          (uint8_t *) private_e_2048,


                    RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(private_e_2048)),
                                          (uint8_t *) secret_2048,
 sizeof(m_2048),
                                          NX_CRYPTO_NULL,
         output,
 sizeof(m_2048),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, plain_2048, sizeof(m_2048));
    assert(0 == err);
}
```

### RSA PKCS1V1.5 Example

This is an example on using the NetX Crypto API to sign and verify input message data. The plain input message is PKCS1V1.5 encoded before signature generation.

PKCS1V15 Example

### HMAC SHA256 Example

This is an example on using the HMAC with SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_hmac_sha256;
/*
C4DA057B81EA740B697FFE1B6EB8591356BA6D5EA7F1B96E4F048030449ACD64E4BB271CB4DCF94937E6
*/
const uint8_t key_256[] =
{
    0xC4, 0xDA, 0x05, 0x7B, 0x81, 0xEA, 0x74, 0x0B, 0x69, 0x7F, 0xFE, 0x1B, 0x6E,
0xB8, 0x59, 0x13,
    0x56, 0xBA, 0x6D, 0x5E, 0xA7, 0xF1, 0xB9, 0x6E, 0x4F, 0x04, 0x80, 0x30, 0x44,
0x9A, 0xCD, 0x64,
    0xE4, 0xBB, 0x27, 0x1C, 0xB4, 0xDC, 0xF9, 0x49, 0x37, 0xE6,
};
/* BDACB6555D294D3AFFC245520116062D98F88D64276BDA593492AE71CFE16E46CABC287CB00DF21D96
066D5856C2224EEF609D4896302540078F3A0EE325F5337E */
const uint8_t plain_256[] =
{
    0xBD, 0xAC, 0xB6, 0x55, 0x5D, 0x29, 0x4D, 0x3A, 0xFF, 0xC2, 0x45, 0x52, 0x01,
0x16, 0x06, 0x2D,
    0x98, 0xF8, 0x8D, 0x64, 0x27, 0x6B, 0xDA, 0x59, 0x34, 0x92, 0xAE, 0x71, 0xCF,
0xE1, 0x6E, 0x46,
    0xCA, 0xBC, 0x28, 0x7C, 0xB0, 0x0D, 0xF2, 0x1D, 0x96, 0x06, 0x6D, 0x58, 0x56,
0xC2, 0x22, 0x4E,
    0xEF, 0x60, 0x9D, 0x48, 0x96, 0x30, 0x25, 0x40, 0x07, 0x8F, 0x3A, 0x0E, 0xE3,
0x25, 0xF5, 0x33,
    0x7E,
};
/* 940F986AC891C9000B72EF0CEC69AB66AF002E3A34EB8A3A5F94484E45C0396C */
const uint8_t secret_256[] =
{
    0x94, 0x0F, 0x98, 0x6A, 0xC8, 0x91, 0xC9, 0x00, 0x0B, 0x72, 0xEF, 0x0C, 0xEC,
0x69, 0xAB, 0x66,
    0xAF, 0x00, 0x2E, 0x3A, 0x34, 0xEB, 0x8A, 0x3A, 0x5F, 0x94, 0x48, 0x4E, 0x45,
0xC0, 0x39, 0x6C,
};
```

```
void netx_secure_crypto_hmac_sha256_example (void)
{
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES] = {0};
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256_HMAC)] = {0};
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256_HMAC);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 init */
    err = _nx_crypto_method_hmac_sha256_init(&crypto_method_hmac_sha256,
                                             (UCHAR *) key_256,




                RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
             &handler,
                                             metadata,
             metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
    err =
       _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                               handler,
                                                &crypto_method_hmac_sha256,
                                               (UCHAR *) key_256,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
                                               NX_CRYPTO_NULL,
                                               0,
                                               NX_CRYPTO_NULL,
                                               NX_CRYPTO_NULL,
                                               0,
```

```
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_UPDATE */
   err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                        handler,
                                           &crypto_method_hmac_sha256,
                                        NX_CRYPTO_NULL,
                                        0,
                                          (UCHAR *) plain_256,
sizeof(plain_256),
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL,
                                        0,
                                        metadata,
                                        metadata_size,
                                        NX_CRYPTO_NULL,
                                        NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
   err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                        handler,
                                           &crypto_method_hmac_sha256,
                                        NX_CRYPTO_NULL,
                                        0,
                                        NX_CRYPTO_NULL,
                                        0,
                                        NX_CRYPTO_NULL,
                                          (UCHAR *) output,
sizeof(output),
                                        metadata,
                                        metadata_size,
```

```
                                            NX_CRYPTO_NULL,
                                            NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Ensure generated HMAC-SHA256 mac matches the expected mac */
    err = (uint32_t) memcmp(output, secret_256, sizeof(secret_256));
    assert(0 == err);
}
```

### 5.2.15.3 Mbed Crypto H/W Acceleration (rm_psa_crypto)

Modules » Security

#### Functions

| | |
|---|---|
| int | mbedtls_platform_setup (mbedtls_platform_context *ctx) |
| void | mbedtls_platform_teardown (mbedtls_platform_context *ctx) |
| fsp_err_t | RM_PSA_CRYPTO_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes, uint32_t *p_num_gen_bytes)<br>Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer. More... |

#### Detailed Description

Hardware acceleration for the mbedCrypto implementation of the Arm PSA Crypto API.

# Overview

*Note*

***The PSA Crypto module does not provide any interfaces to the user. This release uses the mbedTLS version 3.6.2 which conforms to the PSA Crypto API 1.0 specification. Consult the Arm documentation at https://armmbed.github.io/mbed-crypto/psa/#application-programming-interface for further information.*** *FSP 3.0 onward adopts a change by Arm where mbedCrypto has been integrated back to MbedTLS and the term mbedCrypto has been deprecated. The mbedCrypto term in FSP now refers to the crypto portion of the MbedTLS module.*

#### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E51A | RA8M1, RA8D1, RA8T1 |
| RSIP-E50D | RA8P1 |

| | |
|---|---|
| RSIP-E31A | RA4C1 |
| RSIP-E11A | RA4L1 |
| SCE9 | RA6M4, RA4M3, RA4M2, RA6M5 |
| SCE7 | RA6M3, RA6M2, RA6M1, RA6T1 |
| SCE5 | RA4W1, RA4M1 |
| SCE5B | RA6T2 |
| AES Engine | RA2A1, RA2E1, RA2E2, RA2L1 |
| AES Engine 2 | RA2A2 |
| TRNG | RA0E1, RA0E2, RA2L2, RA4E1, RA4E2, RA4T1, RA6E1, RA6E2, RA6T3, RA8E1, RA8E2 |

### Features

| | | AES Engine | AES Engine 2 | SCE5 | SCE5_B | SCE7 | SCE9 | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TRNG | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Hash | SHA224, SHA256 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | SHA384, SHA512 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| | SHA3-(224/256/384/512) | – | – | – | – | – | – | – | – | – | Yes |
| MAC | HMAC-SHA224, HMAC-SHA256 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | HMAC-SHA384, HMAC-SHA512 | – | – | – | – | – | – | – | Yes | Yes | Yes |
| | AES128-CMAC | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES192-CMAC | – | – | – | – | – | – | – | – | Yes | Yes |
| | AES25 | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 6-CMAC | | | | | | | | | | |
| AES | AES128-ECB, CBC, CTR | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES128-XTS | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES128-GCM, GMAC, CCM | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES192-ECB, CBC, CTR | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES192-GCM, GMAC, CCM | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES256-ECB, CBC, CTR | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | AES256-XTS | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | AES256-GCM, GMAC, CCM | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | Key generation - plaintext | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| | Key generation - wrapped | – | – | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| ChaCha | ChaCha20 | – | – | – | – | – | – | – | – | – | Yes |
| | ChaCha20-P | – | – | – | – | – | – | – | – | – | Yes |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | oly1305 | | | | | | | | | | |
| ECC | secp256r1 | – | – | – | – | Yes | Yes | Yes | Yes | Yes | Yes |
| | secp384r1 | – | – | – | – | Yes | Yes | – | Yes | Yes | Yes |
| | secp521r1 | – | – | – | – | – | – | – | – | Yes | Yes |
| | Brainpool256r1 | – | – | – | – | Yes | Yes | – | Yes | Yes | Yes |
| | Brainpool384r1 | – | – | – | – | Yes | Yes | – | Yes | Yes | Yes |
| | Key generation - plaintext | – | – | – | – | Yes | Yes | – | – | – | – |
| | Key generation - wrapped | – | – | – | – | – | Yes | Yes | Yes | Yes | Yes |
| RSA | RSA-1024 | – | – | – | – | – | Yes | – | – | Yes | Yes |
| | RSA-2048 | – | – | – | – | Yes | Yes | – | – | Yes | Yes |
| | RSA-3072 | – | – | – | – | – | (1) | – | – | Yes | Yes |
| | RSA-4096 | – | – | – | – | – | (1) | – | – | Yes | Yes |
| | Key generation - plaintext | – | – | – | – | Yes | – | – | – | – | – |
| | Key generation - wrapped | – | – | – | – | Yes | (2) | – | – | (3) | (3) |

*Note*

*1. Signature verification only*

*2. RSA-1024 and RSA-2048 only*
*3. RSA-2048, RSA-3072, RSA-4096 only*

# Configuration

### Build Time Configurations for mbedCrypto

The following build time configurations are defined in arm/mbedtls/config.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Hardware Acceleration | | | |
| Hardware Acceleration > Key Format | | | |
| AES | MCU Specific Options | | Select AES key formats used |
| ECC | MCU Specific Options | | Select ECC key formats used |
| RSA | MCU Specific Options | | Select RSA key formats used |
| Hardware Acceleration > Hash | | | |
| SHA256/224 | MCU Specific Options | | Defines MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT. |
| SHA512/384 | MCU Specific Options | | Defines MBEDTLS_SHA512_ALT and MBEDTLS_SHA512_PROCESS_ALT. |
| SHA3_224/256/384/512 | MCU Specific Options | | Defines MBEDTLS_SHA3_ALT and MBEDTLS_SHA3_PROCESS_ALT. |
| Hardware Acceleration > Cipher | | | |
| AES | MCU Specific Options | | Defines MBEDTLS_AES_ALT, MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT |
| Hardware Acceleration > Public Key Cryptography (PKC) | | | |
| Hardware Acceleration > Public Key Cryptography (PKC) > RSA 3072 | | | |
| Key Generation | MCU Specific Options | | Enables RSA 3072 Key |

| | | | |
|---|---|---|---|
| | | | Generation. |
| Signing | MCU Specific Options | | Enables RSA 3072 Key Signing. |
| Verification | MCU Specific Options | | Enables RSA 3072 Verify. |
| Hardware Acceleration > Public Key Cryptography (PKC) > RSA 4096 | | | |
| Key Generation | MCU Specific Options | | Enables RSA 4096 Key Generation. |
| Signing | MCU Specific Options | | Enables RSA 4096 Key Signing. |
| Verification | MCU Specific Options | | Enables RSA 4096 Verify. |
| ECC | MCU Specific Options | | Defines MBEDTLS_ECP_ALT |
| ECDSA | MCU Specific Options | | Defines MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT |
| ECDH | MCU Specific Options | | Defines MBEDTLS_ECDH_ALT |
| RSA 1024 | MCU Specific Options | | Defines MBEDTLS_RSA_1024_ALT. RSA 1024 Key Generation, Signing and Verification are also enabled. |
| RSA 2048 | MCU Specific Options | | Defines MBEDTLS_RSA_ALT. RSA 2048 Key Generation, Signing and Verification are also enabled. |
| TRNG | Enabled | Enabled | Defines MBEDTLS_ENTROPY_HARDWARE_ALT. |
| Crypto Engine Initialization | Enabled | Enabled | MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT |
| Platform | | | |
| Platform > Alternate | | | |
| MBEDTLS_PLATFORM_SETBUF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_SETBUF_ALT |
| MBEDTLS_PLATFORM_EXIT_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_EXIT_ALT |
| MBEDTLS_PLATFORM_TIME_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_ALT |

| | | | |
|---|---|---|---|
| MBEDTLS_PLATFORM_FPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FPRINTF_ALT |
| MBEDTLS_PLATFORM_PRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_PRINTF_ALT |
| MBEDTLS_PLATFORM_SNPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_SNPRINTF_ALT |
| MBEDTLS_PLATFORM_VSNPRINTF_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_VSNPRINTF_ALT |
| MBEDTLS_PLATFORM_NV_SEED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_ALT |
| MBEDTLS_PLATFORM_MS_TIME_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_MS_TIME_ALT |
| MBEDTLS_PLATFORM_ZEROIZE_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_ZEROIZE_ALT |
| MBEDTLS_PLATFORM_GMTIME_R_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_GMTIME_R_ALT |
| MBEDTLS_HAVE_ASM | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_ASM |
| MBEDTLS_NO_UDBL_DIVISION | • Define<br>• Undefine | Undefine | MBEDTLS_NO_UDBL_DIVISION |
| MBEDTLS_NO_64BIT_MULTIPLICATION | • Define<br>• Undefine | Undefine | MBEDTLS_NO_64BIT_MULTIPLICATION |
| MBEDTLS_HAVE_SSE2 | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_SSE2 |
| MBEDTLS_HAVE_TIME | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_TIME |
| MBEDTLS_HAVE_TIME_DATE | • Define<br>• Undefine | Undefine | MBEDTLS_HAVE_TIME_DATE |
| MBEDTLS_PLATFORM_MEMORY | • Define<br>• Undefine | Define | MBEDTLS_PLATFORM_MEMORY |
| MBEDTLS_PLATFORM_NO_STD_FUNCTIONS | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NO_STD_FUNCTIONS |
| MBEDTLS_TIMING_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_TIMING_ALT |
| MBEDTLS_NO_PLATFORM_ENTROPY | • Define<br>• Undefine | Define | MBEDTLS_NO_PLATFORM_ENTROPY |
| MBEDTLS_ENTROPY_C | • Define<br>• Undefine | Define | MBEDTLS_ENTROPY_C |
| MBEDTLS_PLATFORM_C | • Define<br>• Undefine | Define | MBEDTLS_PLATFORM_C |
| MBEDTLS_PLATFORM_STD_CALLOC | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_CALLOC |

| | | | |
|---|---|---|---|
| MBEDTLS_PLATFORM_STD_CALLOC value | Manual Entry | calloc | MBEDTLS_PLATFORM_STD_CALLOC value |
| MBEDTLS_PLATFORM_STD_FREE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_FREE |
| MBEDTLS_PLATFORM_STD_FREE value | Manual Entry | free | MBEDTLS_PLATFORM_STD_FREE value |
| MBEDTLS_PLATFORM_STD_SETBUF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_SETBUF |
| MBEDTLS_PLATFORM_STD_SETBUF value | Manual Entry | setbuf | MBEDTLS_PLATFORM_STD_SETBUF value |
| MBEDTLS_PLATFORM_STD_EXIT | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT |
| MBEDTLS_PLATFORM_STD_EXIT value | Manual Entry | exit | MBEDTLS_PLATFORM_STD_EXIT value |
| MBEDTLS_PLATFORM_STD_TIME | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_TIME |
| MBEDTLS_PLATFORM_STD_TIME value | Manual Entry | time | MBEDTLS_PLATFORM_STD_TIME value |
| MBEDTLS_PLATFORM_STD_FPRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_FPRINTF |
| MBEDTLS_PLATFORM_STD_FPRINTF value | Manual Entry | fprintf | MBEDTLS_PLATFORM_STD_FPRINTF value |
| MBEDTLS_PLATFORM_STD_PRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_PRINTF |
| MBEDTLS_PLATFORM_STD_PRINTF value | Manual Entry | printf | MBEDTLS_PLATFORM_STD_PRINTF value |
| MBEDTLS_PLATFORM_STD_SNPRINTF | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_SNPRINTF |
| MBEDTLS_PLATFORM_STD_SNPRINTF value | Manual Entry | snprintf | MBEDTLS_PLATFORM_STD_SNPRINTF value |
| MBEDTLS_PLATFORM_STD_EXIT_SUCCESS | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT_SUCCESS |
| MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value | Manual Entry | 0 | MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value |
| MBEDTLS_PLATFORM_STD_EXIT_FAILURE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_EXIT_FAILURE |
| MBEDTLS_PLATFORM_STD_EXIT_FAILURE value | Manual Entry | 1 | MBEDTLS_PLATFORM_STD_EXIT_FAILURE value |
| MBEDTLS_PLATFORM_STD_NV_SEED_READ | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_READ |
| MBEDTLS_PLATFORM_S | Manual Entry | mbedtls_platform_std_ | MBEDTLS_PLATFORM_S |

| | | | |
|---|---|---|---|
| TD_NV_SEED_READ value | | nv_seed_read | TD_NV_SEED_READ value |
| MBEDTLS_PLATFORM_STD_NV_SEED_WRITE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_WRITE |
| MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value | Manual Entry | mbedtls_platform_std_nv_seed_write | MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value |
| MBEDTLS_PLATFORM_STD_NV_SEED_FILE | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_STD_NV_SEED_FILE |
| MBEDTLS_PLATFORM_STD_NV_SEED_FILE value | Manual Entry | | MBEDTLS_PLATFORM_STD_NV_SEED_FILE value |
| MBEDTLS_PLATFORM_CALLOC_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_CALLOC_MACRO |
| MBEDTLS_PLATFORM_CALLOC_MACRO value | Manual Entry | calloc | MBEDTLS_PLATFORM_CALLOC_MACRO value |
| MBEDTLS_PLATFORM_FREE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FREE_MACRO |
| MBEDTLS_PLATFORM_FREE_MACRO value | Manual Entry | free | MBEDTLS_PLATFORM_FREE_MACRO value |
| MBEDTLS_PLATFORM_EXIT_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_EXIT_MACRO |
| MBEDTLS_PLATFORM_EXIT_MACRO value | Manual Entry | exit | MBEDTLS_PLATFORM_EXIT_MACRO value |
| MBEDTLS_PLATFORM_SETBUF_MACRO | • Define<br>• Undefine | Define | MBEDTLS_PLATFORM_SETBUF_MACRO |
| MBEDTLS_PLATFORM_SETBUF_MACRO value | Manual Entry | dummy_setbuf | MBEDTLS_PLATFORM_SETBUF_MACRO value |
| MBEDTLS_PLATFORM_TIME_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_MACRO |
| MBEDTLS_PLATFORM_TIME_MACRO value | Manual Entry | time | MBEDTLS_PLATFORM_TIME_MACRO value |
| MBEDTLS_PLATFORM_TIME_TYPE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_TIME_TYPE_MACRO |
| MBEDTLS_PLATFORM_TIME_TYPE_MACRO value | Manual Entry | time_t | MBEDTLS_PLATFORM_TIME_TYPE_MACRO value |
| MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO |
| MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO value | Manual Entry | int64_t | MBEDTLS_PLATFORM_MS_TIME_TYPE_MACRO value |
| MBEDTLS_PRINTF_MS_T | • Define | Undefine | MBEDTLS_PRINTF_MS_T |

| | | | |
|---|---|---|---|
| IME | • Undefine | | IME |
| MBEDTLS_PRINTF_MS_TIME value | Manual Entry | PRId64 | MBEDTLS_PRINTF_MS_TIME value |
| MBEDTLS_PLATFORM_FPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_FPRINTF_MACRO |
| MBEDTLS_PLATFORM_FPRINTF_MACRO value | Manual Entry | fprintf | MBEDTLS_PLATFORM_FPRINTF_MACRO value |
| MBEDTLS_PLATFORM_PRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_PRINTF_MACRO |
| MBEDTLS_PLATFORM_PRINTF_MACRO value | Manual Entry | printf | MBEDTLS_PLATFORM_PRINTF_MACRO value |
| MBEDTLS_PLATFORM_SNPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_SNPRINTF_MACRO |
| MBEDTLS_PLATFORM_SNPRINTF_MACRO value | Manual Entry | snprintf | MBEDTLS_PLATFORM_SNPRINTF_MACRO value |
| MBEDTLS_PLATFORM_VSNPRINTF_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_VSNPRINTF_MACRO |
| MBEDTLS_PLATFORM_VSNPRINTF_MACRO value | Manual Entry | vsnprintf | MBEDTLS_PLATFORM_VSNPRINTF_MACRO value |
| MBEDTLS_PLATFORM_NV_SEED_READ_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_READ_MACRO |
| MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value | Manual Entry | mbedtls_platform_std_nv_seed_read | MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value |
| MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO | • Define<br>• Undefine | Undefine | MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO |
| MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value | Manual Entry | mbedtls_platform_std_nv_seed_write | MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value |
| General | | | |
| MBEDTLS_PSA_CRYPTO_DRIVERS | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_DRIVERS |
| MBEDTLS_DEPRECATED_WARNING | • Define<br>• Undefine | Undefine | MBEDTLS_DEPRECATED_WARNING |
| MBEDTLS_DEPRECATED_REMOVED | • Define<br>• Undefine | Define | MBEDTLS_DEPRECATED_REMOVED |
| MBEDTLS_CHECK_RETURN_WARNING | • Define<br>• Undefine | Undefine | MBEDTLS_CHECK_RETURN_WARNING |
| MBEDTLS_ERROR_STRERROR_DUMMY | • Define<br>• Undefine | Define | MBEDTLS_ERROR_STRERROR_DUMMY |
| MBEDTLS_MEMORY_DE | • Define | Undefine | MBEDTLS_MEMORY_DE |

| | | | |
|---|---|---|---|
| BUG | • Undefine | | BUG |
| MBEDTLS_MEMORY_BACKTRACE | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_BACKTRACE |
| MBEDTLS_PSA_CRYPTO_CLIENT | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_CLIENT |
| MBEDTLS_PSA_CRYPTO_SPM | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_SPM |
| MBEDTLS_PSA_KEY_STORE_DYNAMIC | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_KEY_STORE_DYNAMIC |
| MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS |
| MBEDTLS_SELF_TEST | • Define<br>• Undefine | Undefine | MBEDTLS_SELF_TEST |
| MBEDTLS_THREADING_ALT | • Define<br>• Undefine | Define | MBEDTLS_THREADING_ALT |
| MBEDTLS_THREADING_PTHREAD | • Define<br>• Undefine | Undefine | MBEDTLS_THREADING_PTHREAD |
| MBEDTLS_USE_PSA_CRYPTO | Undefine | Undefine | MBEDTLS_USE_PSA_CRYPTO |
| MBEDTLS_VERSION_FEATURES | • Define<br>• Undefine | Define | MBEDTLS_VERSION_FEATURES |
| MBEDTLS_ERROR_C | • Define<br>• Undefine | Define | MBEDTLS_ERROR_C |
| MBEDTLS_MEMORY_BUFFER_ALLOC_C | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_BUFFER_ALLOC_C |
| MBEDTLS_PSA_CRYPTO_C | • Define<br>• Undefine | Define | MBEDTLS_PSA_CRYPTO_C |
| MBEDTLS_PSA_CRYPTO_SE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_SE_C |
| MBEDTLS_THREADING_C | • Define<br>• Undefine | Define | MBEDTLS_THREADING_C |
| MBEDTLS_TIMING_C | • Define<br>• Undefine | Undefine | MBEDTLS_TIMING_C |
| MBEDTLS_VERSION_C | • Define<br>• Undefine | Define | MBEDTLS_VERSION_C |
| MBEDTLS_MEMORY_ALIGN_MULTIPLE | • Define<br>• Undefine | Undefine | MBEDTLS_MEMORY_ALIGN_MULTIPLE |
| MBEDTLS_MEMORY_ALIGN_MULTIPLE value | Manual Entry | 4 | MBEDTLS_MEMORY_ALIGN_MULTIPLE value |
| MBEDTLS_CHECK_RETURN | • Define<br>• Undefine | Define | MBEDTLS_CHECK_RETURN |
| MBEDTLS_IGNORE_RET | • Define | Undefine | MBEDTLS_IGNORE_RET |

| | | | |
|---|---|---|---|
| URN | • Undefine | | URN |
| MBEDTLS_PSA_CRYPTO_CONFIG | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_CONFIG |
| Cipher | | | |
| Cipher > Alternate | | | |
| MBEDTLS_ARIA_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ARIA_ALT |
| MBEDTLS_CAMELLIA_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_ALT |
| MBEDTLS_CCM_ALT | MCU Specific Options | | MBEDTLS_CCM_ALT |
| MBEDTLS_CHACHA20_ALT | MCU Specific Options | | MBEDTLS_CHACHA20_ALT |
| MBEDTLS_CHACHAPOLY_ALT | MCU Specific Options | | MBEDTLS_CHACHAPOLY_ALT |
| MBEDTLS_CMAC_ALT | MCU Specific Options | | MBEDTLS_CMAC_ALT |
| MBEDTLS_DES_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_ALT |
| MBEDTLS_GCM_ALT | MCU Specific Options | | MBEDTLS_GCM_ALT |
| MBEDTLS_NIST_KW_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_NIST_KW_ALT |
| MBEDTLS_DES_SETKEY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_SETKEY_ALT |
| MBEDTLS_DES_CRYPT_ECB_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES_CRYPT_ECB_ALT |
| MBEDTLS_DES3_CRYPT_ECB_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DES3_CRYPT_ECB_ALT |
| Cipher > AES | | | |
| MBEDTLS_AES_ROM_TABLES | • Define<br>• Undefine | Undefine | MBEDTLS_AES_ROM_TABLES |
| MBEDTLS_AES_FEWER_TABLES | • Define<br>• Undefine | Undefine | MBEDTLS_AES_FEWER_TABLES |
| MBEDTLS_AES_ONLY_128_BIT_KEY_LENGTH | • Define<br>• Undefine | Undefine | MBEDTLS_AES_ONLY_128_BIT_KEY_LENGTH |
| MBEDTLS_CAMELLIA_SMALL_MEMORY | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_SMALL_MEMORY |
| MBEDTLS_CIPHER_MODE_CBC | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_MODE_CBC |
| MBEDTLS_CIPHER_MODE_CFB | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_MODE_CFB |
| MBEDTLS_CIPHER_MOD | • Define | Define | MBEDTLS_CIPHER_MOD |

| | | | |
|---|---|---|---|
| E_CTR | • Undefine | | E_CTR |
| MBEDTLS_CIPHER_MODE_OFB | • Define<br>• Undefine | Undefine | MBEDTLS_CIPHER_MODE_OFB |
| MBEDTLS_CIPHER_MODE_XTS | • Define<br>• Undefine | Undefine | MBEDTLS_CIPHER_MODE_XTS |
| MBEDTLS_CIPHER_NULL_CIPHER | • Define<br>• Undefine | Undefine | MBEDTLS_CIPHER_NULL_CIPHER |
| MBEDTLS_CIPHER_PADDING_PKCS7 | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_PKCS7 |
| MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS |
| MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN |
| MBEDTLS_CIPHER_PADDING_ZEROS | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_PADDING_ZEROS |
| MBEDTLS_AES_C | Define | Define | MBEDTLS_AES_C |
| MBEDTLS_BLOCK_CIPHER_NO_DECRYPT | • Define<br>• Undefine | Undefine | MBEDTLS_BLOCK_CIPHER_NO_DECRYPT |
| MBEDTLS_CAMELLIA_C | • Define<br>• Undefine | Undefine | MBEDTLS_CAMELLIA_C |
| MBEDTLS_ARIA_C | • Define<br>• Undefine | Undefine | MBEDTLS_ARIA_C |
| MBEDTLS_CCM_C | • Define<br>• Undefine | Define | MBEDTLS_CCM_C |
| MBEDTLS_CHACHA20_C | • Define<br>• Undefine | Undefine | MBEDTLS_CHACHA20_C |
| MBEDTLS_CHACHAPOLY_C | • Define<br>• Undefine | Undefine | MBEDTLS_CHACHAPOLY_C |
| MBEDTLS_CIPHER_C | • Define<br>• Undefine | Define | MBEDTLS_CIPHER_C |
| MBEDTLS_DES_C | • Define<br>• Undefine | Undefine | MBEDTLS_DES_C |
| MBEDTLS_GCM_C | • Define<br>• Undefine | Define | MBEDTLS_GCM_C |
| MBEDTLS_GCM_LARGE_TABLE | • Define<br>• Undefine | Undefine | MBEDTLS_GCM_LARGE_TABLE |
| MBEDTLS_NIST_KW_C | • Define<br>• Undefine | Undefine | MBEDTLS_NIST_KW_C |

Public Key Cryptography (PKC)

Public Key Cryptography (PKC) > DHM

Public Key Cryptography (PKC) > DHM > Alternate

| | | | |
|---|---|---|---|
| MBEDTLS_DHM_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_DHM_ALT |
| MBEDTLS_DHM_C | • Define<br>• Undefine | Undefine | MBEDTLS_DHM_C |

Public Key Cryptography (PKC) > ECC

Public Key Cryptography (PKC) > ECC > Alternate

| | | | |
|---|---|---|---|
| MBEDTLS_ECJPAKE_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECJPAKE_ALT |
| MBEDTLS_ECDSA_GEN KEY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDSA_GEN KEY_ALT |
| MBEDTLS_ECP_INTERN AL_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_INTERN AL_ALT |
| MBEDTLS_ECP_RANDO MIZE_JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RANDO MIZE_JAC_ALT |
| MBEDTLS_ECP_ADD_MI XED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_ADD_MI XED_ALT |
| MBEDTLS_ECP_DOUBLE _JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DOUBLE _JAC_ALT |
| MBEDTLS_ECP_NORMA LIZE_JAC_MANY_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMA LIZE_JAC_MANY_ALT |
| MBEDTLS_ECP_NORMA LIZE_JAC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMA LIZE_JAC_ALT |
| MBEDTLS_ECP_DOUBLE _ADD_MXZ_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DOUBLE _ADD_MXZ_ALT |
| MBEDTLS_ECP_RANDO MIZE_MXZ_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RANDO MIZE_MXZ_ALT |
| MBEDTLS_ECP_NORMA LIZE_MXZ_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NORMA LIZE_MXZ_ALT |

Public Key Cryptography (PKC) > ECC > Curves

| | | | |
|---|---|---|---|
| MBEDTLS_ECP_DP_SEC P192R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SEC P192R1_ENABLED |
| MBEDTLS_ECP_DP_SEC P224R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SEC P224R1_ENABLED |
| MBEDTLS_ECP_DP_SEC P256R1_ENABLED | • Define<br>• Undefine | Define | MBEDTLS_ECP_DP_SEC P256R1_ENABLED |
| MBEDTLS_ECP_DP_SEC P384R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SEC P384R1_ENABLED |
| MBEDTLS_ECP_DP_SEC P521R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SEC P521R1_ENABLED |
| MBEDTLS_ECP_DP_SEC P192K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SEC P192K1_ENABLED |

| | | | |
|---|---|---|---|
| MBEDTLS_ECP_DP_SECP224K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP224K1_ENABLED |
| MBEDTLS_ECP_DP_SECP256K1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_SECP256K1_ENABLED |
| MBEDTLS_ECP_DP_BP256R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_BP256R1_ENABLED |
| MBEDTLS_ECP_DP_BP384R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_BP384R1_ENABLED |
| MBEDTLS_ECP_DP_BP512R1_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_BP512R1_ENABLED |
| MBEDTLS_ECP_DP_CURVE25519_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_CURVE25519_ENABLED |
| MBEDTLS_ECP_DP_CURVE448_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_DP_CURVE448_ENABLED |
| MBEDTLS_ECDH_GEN_PUBLIC_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_GEN_PUBLIC_ALT |
| MBEDTLS_ECDH_COMPUTE_SHARED_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_COMPUTE_SHARED_ALT |
| MBEDTLS_ECP_NO_FALLBACK | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NO_FALLBACK |
| MBEDTLS_ECP_NIST_OPTIM | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_NIST_OPTIM |
| MBEDTLS_ECP_RESTARTABLE | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_RESTARTABLE |
| MBEDTLS_ECDSA_DETERMINISTIC | • Define<br>• Undefine | Undefine | MBEDTLS_ECDSA_DETERMINISTIC |
| MBEDTLS_PK_PARSE_EC_COMPRESSED | • Define<br>• Undefine | Undefine | MBEDTLS_PK_PARSE_EC_COMPRESSED |
| MBEDTLS_PK_PARSE_EC_EXTENDED | • Define<br>• Undefine | Undefine | MBEDTLS_PK_PARSE_EC_EXTENDED |
| MBEDTLS_ECDH_C | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_C |
| MBEDTLS_ECDSA_C | • Define<br>• Undefine | Define | MBEDTLS_ECDSA_C |
| MBEDTLS_ECP_C | • Define<br>• Undefine | Define | MBEDTLS_ECP_C |
| MBEDTLS_ECJPAKE_C | • Define<br>• Undefine | Undefine | MBEDTLS_ECJPAKE_C |
| MBEDTLS_ECP_WINDOW_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_WINDOW_SIZE |
| MBEDTLS_ECP_WINDOW_SIZE value | Manual Entry | 6 | MBEDTLS_ECP_WINDOW_SIZE value |

| | | | |
|---|---|---|---|
| MBEDTLS_ECP_FIXED_POINT_OPTIM | • Define<br>• Undefine | Undefine | MBEDTLS_ECP_FIXED_POINT_OPTIM |
| MBEDTLS_ECP_FIXED_POINT_OPTIM value | Manual Entry | 1 | MBEDTLS_ECP_FIXED_POINT_OPTIM value |
| MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED | • Define<br>• Undefine | Undefine | MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED |
| Public Key Cryptography (PKC) > RSA | | | |
| MBEDTLS_PK_RSA_ALT_SUPPORT | • Define<br>• Undefine | Undefine | MBEDTLS_PK_RSA_ALT_SUPPORT |
| MBEDTLS_RSA_NO_CRT | • Define<br>• Undefine | Define | MBEDTLS_RSA_NO_CRT |
| MBEDTLS_RSA_C | • Define<br>• Undefine | Define | MBEDTLS_RSA_C |
| MBEDTLS_RSA_GEN_KEY_MIN_BITS | • Define<br>• Undefine | Undefine | MBEDTLS_RSA_GEN_KEY_MIN_BITS |
| MBEDTLS_RSA_GEN_KEY_MIN_BITS value | Manual Entry | 1024 | MBEDTLS_RSA_GEN_KEY_MIN_BITS value |
| MBEDTLS_GENPRIME | • Define<br>• Undefine | Define | MBEDTLS_GENPRIME |
| MBEDTLS_PKCS1_V15 | • Define<br>• Undefine | Define | MBEDTLS_PKCS1_V15 |
| MBEDTLS_PKCS1_V21 | • Define<br>• Undefine | Define | MBEDTLS_PKCS1_V21 |
| MBEDTLS_ASN1_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_ASN1_PARSE_C |
| MBEDTLS_ASN1_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_ASN1_WRITE_C |
| MBEDTLS_BASE64_C | • Define<br>• Undefine | Define | MBEDTLS_BASE64_C |
| MBEDTLS_BIGNUM_C | • Define<br>• Undefine | Define | MBEDTLS_BIGNUM_C |
| MBEDTLS_LMS_C | • Define<br>• Undefine | Define | MBEDTLS_LMS_C |
| MBEDTLS_LMS_PRIVATE | • Define<br>• Undefine | Undefine | MBEDTLS_LMS_PRIVATE |
| MBEDTLS_OID_C | • Define<br>• Undefine | Define | MBEDTLS_OID_C |
| MBEDTLS_PEM_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_PEM_PARSE_C |
| MBEDTLS_PEM_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_PEM_WRITE_C |
| MBEDTLS_PK_C | • Define | Define | MBEDTLS_PK_C |

| | | | |
|---|---|---|---|
| | • Undefine | | |
| MBEDTLS_PK_PARSE_C | • Define<br>• Undefine | Define | MBEDTLS_PK_PARSE_C |
| MBEDTLS_PK_WRITE_C | • Define<br>• Undefine | Define | MBEDTLS_PK_WRITE_C |
| MBEDTLS_PKCS5_C | • Define<br>• Undefine | Define | MBEDTLS_PKCS5_C |
| MBEDTLS_PKCS7_C | • Define<br>• Undefine | Undefine | MBEDTLS_PKCS7_C |
| MBEDTLS_PKCS12_C | • Define<br>• Undefine | Define | MBEDTLS_PKCS12_C |
| MBEDTLS_MPI_WINDOW_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_MPI_WINDOW_SIZE |
| MBEDTLS_MPI_WINDOW_SIZE value | Manual Entry | 6 | MBEDTLS_MPI_WINDOW_SIZE value |
| MBEDTLS_MPI_MAX_SIZE | • Define<br>• Undefine | Undefine | MBEDTLS_MPI_MAX_SIZE |
| MBEDTLS_MPI_MAX_SIZE value | Manual Entry | 1024 | MBEDTLS_MPI_MAX_SIZE value |
| Hash | | | |
| Hash > Alternate | | | |
| MBEDTLS_MD5_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD5_ALT |
| MBEDTLS_RIPEMD160_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_RIPEMD160_ALT |
| MBEDTLS_SHA1_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA1_ALT |
| MBEDTLS_MD5_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_MD5_PROCESS_ALT |
| MBEDTLS_RIPEMD160_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_RIPEMD160_PROCESS_ALT |
| MBEDTLS_SHA1_PROCESS_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_SHA1_PROCESS_ALT |
| MBEDTLS_SHA256_SMALLER | • Define<br>• Undefine | Undefine | MBEDTLS_SHA256_SMALLER |
| MBEDTLS_SHA512_SMALLER | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_SMALLER |
| MBEDTLS_MD_C | • Define<br>• Undefine | Define | MBEDTLS_MD_C |
| MBEDTLS_MD5_C | • Define<br>• Undefine | Define | MBEDTLS_MD5_C |

| | | | |
|---|---|---|---|
| MBEDTLS_RIPEMD160_C | • Define<br>• Undefine | Undefine | MBEDTLS_RIPEMD160_C |
| MBEDTLS_SHA1_C | • Define<br>• Undefine | Define | MBEDTLS_SHA1_C |
| MBEDTLS_SHA3_C | • Define<br>• Undefine | Undefine | MBEDTLS_SHA3_C |
| MBEDTLS_SHA224_C | • Define<br>• Undefine | Define | MBEDTLS_SHA224_C |
| MBEDTLS_SHA256_C | • Define<br>• Undefine | Define | MBEDTLS_SHA256_C |
| MBEDTLS_SHA384_C | • Define<br>• Undefine | Undefine | MBEDTLS_SHA384_C |
| MBEDTLS_SHA512_C | • Define<br>• Undefine | Undefine | MBEDTLS_SHA512_C |
| Message Authentication Code (MAC) | | | |
| Message Authentication Code (MAC) > Alternate | | | |
| MBEDTLS_POLY1305_ALT | • Define<br>• Undefine | Undefine | MBEDTLS_POLY1305_ALT |
| MBEDTLS_CMAC_C | • Define<br>• Undefine | Undefine | MBEDTLS_CMAC_C |
| MBEDTLS_HKDF_C | • Define<br>• Undefine | Define | MBEDTLS_HKDF_C |
| MBEDTLS_HMAC_DRBG_C | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_C |
| MBEDTLS_POLY1305_C | • Define<br>• Undefine | Undefine | MBEDTLS_POLY1305_C |
| Storage | | | |
| MBEDTLS_FS_IO | • Define<br>• Undefine | Undefine | MBEDTLS_FS_IO |
| MBEDTLS_PSA_CRYPTO_STORAGE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_STORAGE_C |
| MBEDTLS_PSA_ITS_FILE_C | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_ITS_FILE_C |
| RNG | | | |
| MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES | • Define<br>• Undefine | Undefine | MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES |
| MBEDTLS_ENTROPY_FORCE_SHA256 | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_FORCE_SHA256 |
| MBEDTLS_ENTROPY_NV_SEED | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_NV_SEED |
| MBEDTLS_PSA_CRYPTO | • Define | Undefine | MBEDTLS_PSA_CRYPTO |

| | | | |
|---|---|---|---|
| _EXTERNAL_RNG | • Undefine | | _EXTERNAL_RNG |
| MBEDTLS_PSA_INJECT_ENTROPY | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_INJECT_ENTROPY |
| MBEDTLS_CTR_DRBG_C | • Define<br>• Undefine | Define | MBEDTLS_CTR_DRBG_C |
| MBEDTLS_CTR_DRBG_C_ALT | Define | Define | MBEDTLS_CTR_DRBG_C_ALT |
| MBEDTLS_CTR_DRBG_ENTROPY_LEN | • Define<br>• Undefine | Undefine | RNG\|MBEDTLS_CTR_DRBG_ENTROPY_LEN |
| MBEDTLS_CTR_DRBG_ENTROPY_LEN value | Manual Entry | 48 | RNG value\|MBEDTLS_CTR_DRBG_ENTROPY_LEN |
| MBEDTLS_CTR_DRBG_RESEED_INTERVAL | • Define<br>• Undefine | Undefine | RNG\|MBEDTLS_CTR_DRBG_RESEED_INTERVAL |
| MBEDTLS_CTR_DRBG_RESEED_INTERVAL value | Manual Entry | 10000 | RNG value\|MBEDTLS_CTR_DRBG_RESEED_INTERVAL |
| MBEDTLS_CTR_DRBG_MAX_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_INPUT |
| MBEDTLS_CTR_DRBG_MAX_INPUT value | Manual Entry | 256 | MBEDTLS_CTR_DRBG_MAX_INPUT value |
| MBEDTLS_CTR_DRBG_MAX_REQUEST | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_REQUEST |
| MBEDTLS_CTR_DRBG_MAX_REQUEST value | Manual Entry | 1024 | MBEDTLS_CTR_DRBG_MAX_REQUEST value |
| MBEDTLS_CTR_DRBG_MAX_SEED_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_MAX_SEED_INPUT |
| MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value | Manual Entry | 384 | MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value |
| MBEDTLS_CTR_DRBG_USE_128_BIT_KEY | • Define<br>• Undefine | Undefine | MBEDTLS_CTR_DRBG_USE_128_BIT_KEY |
| MBEDTLS_HMAC_DRBG_RESEED_INTERVAL | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_RESEED_INTERVAL |
| MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value | Manual Entry | 10000 | MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value |
| MBEDTLS_HMAC_DRBG_MAX_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_INPUT |
| MBEDTLS_HMAC_DRBG_MAX_INPUT value | Manual Entry | 256 | MBEDTLS_HMAC_DRBG_MAX_INPUT value |
| MBEDTLS_HMAC_DRBG_MAX_REQUEST | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_REQUEST |

| | | | |
|---|---|---|---|
| MBEDTLS_HMAC_DRBG_MAX_REQUEST value | Manual Entry | 1024 | MBEDTLS_HMAC_DRBG_MAX_REQUEST value |
| MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT | • Define<br>• Undefine | Undefine | MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT |
| MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value | Manual Entry | 384 | MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value |
| MBEDTLS_ENTROPY_MAX_SOURCES | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MAX_SOURCES |
| MBEDTLS_ENTROPY_MAX_SOURCES value | Manual Entry | 20 | MBEDTLS_ENTROPY_MAX_SOURCES value |
| MBEDTLS_ENTROPY_MAX_GATHER | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MAX_GATHER |
| MBEDTLS_ENTROPY_MAX_GATHER value | Manual Entry | 128 | MBEDTLS_ENTROPY_MAX_GATHER value |
| MBEDTLS_ENTROPY_MIN_HARDWARE | • Define<br>• Undefine | Undefine | MBEDTLS_ENTROPY_MIN_HARDWARE |
| MBEDTLS_ENTROPY_MIN_HARDWARE value | Manual Entry | 32 | MBEDTLS_ENTROPY_MIN_HARDWARE value |
| Key Configuration | | | |
| MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER |
| MBEDTLS_PSA_CRYPTO_BUILTIN_KEYS | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_CRYPTO_BUILTIN_KEYS |
| PSA_CRYPTO_DRIVER_TFM_BUILTIN_KEY_LOADER | • Define<br>• Undefine | Undefine | PSA_CRYPTO_DRIVER_TFM_BUILTIN_KEY_LOADER |
| MBEDTLS_PSA_KEY_SLOT_COUNT | • Define<br>• Undefine | Undefine | MBEDTLS_PSA_KEY_SLOT_COUNT |
| MBEDTLS_PSA_KEY_SLOT_COUNT value | Manual Entry | 32 | MBEDTLS_PSA_KEY_SLOT_COUNT value |
| Post Quantum Cryptography (PQC) | | | |
| MBEDTLS_MLKEM_C | • Define<br>• Undefine | Undefine | MBEDTLS_MLKEM_C |
| MBEDTLS_ML_DSA_C | • Define<br>• Undefine | Undefine | MBEDTLS_ML_DSA_C |
| HMAC | | | |
| MBEDTLS_PSA_HMAC_DRBG_MD_TYPE | • Define<br>• Undefine | Define | MBEDTLS_PSA_HMAC_DRBG_MD_TYPE |

**SHA256 Configuration**

To enable hardware acceleration for the SHA256/224 calculation, the macro MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT must be defined in the configuration file. By default SHA256 is enabled. SHA256 can be disabled, but SHA512 then needs to be enabled (software version) because the PSA implementation uses it for the entropy accumulator. This can be done using the RA Configuration editor.

### AES Configuration

To enable hardware acceleration for the AES128/256 operation, the macro MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT must be defined in the configuration file. By default AES is enabled. AES cannot be disabled because the PSA implementation requires it for the CTR_DRBG random number generator. This can be done using the RA Configuration editor.

*Note*

***Only AES XTS 128 is currently supported. RA2 devices support acceleration for ECB part alone, while other devices support full AES XTS hardware acceleration.***

### ECC Configuration

To enable hardware acceleration for the ECC Key Generation operation, the macro MBEDTLS_ECP_ALT must be defined in the configuration file. For ECDSA, the macros MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT must be defined. By default ECC, ECDSA and ECDHE are enabled. To disable ECC, undefine MBEDTLS_ECP_C, MBEDTLS_ECDSA_C and MBEDTLS_ECDH_C. This can be done using the RA Configuration editor.

### RSA Configuration

To enable hardware acceleration for the RSA2048 operation, the macro MBEDTLS_RSA_ALT must be defined in the configuration file. By default RSA is enabled. To disable RSA, undefine MBEDTLS_RSA_C, MBEDTLS_PK_C, MBEDTLS_PK_PARSE_C, MBEDTLS_PK_WRITE_C. This can be done using the RA Configuration editor.

### Post Quantum Cryptography (PQC) Configuration

### ML-KEM Configuration

To enable software only PQC ML-KEM key generation, encapsulation, and decapsulation operations, the macro MBEDTLS_MLKEM_C must be defined in the configuration file. By default, PQC ML-KEM is disabled as the PQC stack is optional. This can be done using the RA Configuration editor.

### ML-DSA Configuration

To enable software only PQC ML-DSA key generation, signing, and verification operations, the macro MBEDTLS_ML_DSA_C must be defined in the configuration file. By default, PQC ML-DSA is disabled as the PQC stack is optional. This can be done using the RA Configuration editor.

### Wrapped Key Usage

To use the Secure Crypto Engine to generate and use wrapped keys, use PSA_KEY_TYPE_AES_WRAPPED or PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(curve) or PSA_KEY_TYPE_RSA_KEY_PAIR when setting the key type attribute. Setting the key's type attribute using this value will cause the SCE to use wrapped key mode for all operations related to that key. The user can use the export functionality to save the wrapped keys to user ROM and import it later for usage. This mode requires that Wrapped Key functionality for the algorithm is enabled in the

project configuration.

*Note*

***On the SCE9 devices, only the RSA public key can be exported. A file system must be used to store the internally generated private key.***

### Persistent Key Storage

Persistent key storage can be enabled by defining MBEDTLS_FS_IO, MBEDTLS_PSA_CRYPTO_STORAGE_C, and MBEDTLS_PSA_ITS_FILE_C. The key lifetime must also be specified as PSA_KEY_LIFETIME_PERSISTENT. A lower level storage module must be added in the RA Configuration editor and initialized in the code before generating persistent keys. Persistent storage supports the use of plaintext and vendor keys. Refer to the lower level storage module documentation for information on how it should be initialized. To generate a persistent key the key must be assigned a unique id prior to calling generate using the psa_set_key_id api.

```
if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
   {
/* Set the id to a positive integer. */
      psa_set_key_id(&attributes, (psa_key_id_t) 5);
   }
```

### Platform Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT must be defined in the configuration file. This enables code that will initialize the SCE. Parameter checking (General|MBEDTLS_CHECK_PARAMS) is enabled by default. To reduce code size, disable parameter checking.

### Random Number Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_ENTROPY_HARDWARE_ALT must be defined in the configuration file. This enables using the TRNG as an entropy source. None of the other cryptographic operations (even in software only mode) will work without this feature.

## Usage Notes

### Hardware Initialization

mbedtls_platform_setup() must be invoked before using the PSA Crypto API to ensure that the SCE peripheral is initialized.

### Memory Usage

In general, depending on the mbedCrypto features being used a heap size of 0x1000 to 0x5000 bytes is required. The total allocated heap should be the **sum** of the heap requirements of the individual algorithms:

| Algorithm | Required Heap (bytes) |
|---|---|

| | |
|---|---|
| SHA256/224 | None |
| AES | 0x200 |
| Hardware ECC | 0x400 |
| Software ECC | 0x1800 |
| RSA | 0x1500 |

A minimum stack of 0x1000 is required where the module is used. This is either the main stack in a bare metal application or the task stack of the task used for crypto operations.

**Limitations**

- Only little endian mode is supported.

**Stdio Buffering**

The MBEDTLS_PLATFORM_SETBUF_MACRO was introduced in mbedTLS 3.2.1 to prevent stdio read/write functions from buffering stream data to reduce the likelihood of key leakage by setting the buffer argument in setbuf() to NULL. FSP uses a dummy_setbuf() function in rm_psa_crypto.c to prevent build errors; since FSP uses LittleFS by default (where the usage of a buffer is mandatory) this function does not perform any action. Setting the cache size in LittleFS to the minimum supported by the Data Flash (4) can minimize but not remove the likelihood of key data leakage. The dummy function can be replaced with a user-defined function by defining a different value for MBEDTLS_PLATFORM_SETBUF_MACRO_value in the FSP configurator.

**Post Quantum Cryptography (PQC)**

- The PSA API Specificiation v1.3 supports ML-KEM and ML-DSA however MbedTLS currently does not provide an implementation. The implementation in the FSP release is thus likely to change and may cause backward compatibility issues when MbedTLS officially supports it.
- ASN.1 encoding not implemented. Keys are currently implemented in raw bytes.

## ML-DSA

- The FIPS204 Standard allows the user to provide an optional context string for ML-DSA signature generation and verification. The current implementation does not allow the user to provide such a string. Internally this field is set to a value of 0 with a length of 0.
- The data format for import and export of the key pair is a byte array of the 32 byte seed value.
- Only ML-DSA-44 and ML-DSA-65 supported. ML-DSA-87 not supported.
- Only PSA_ALG_HASH_ML_DSA supported. PSA_ALG_ML_DSA requires PSA to offer a SHAKE-256 implementation.
- The current implementation of ML-DSA is too big for devices with less than 64KB of RAM.
- There is a known runtime issue when IAR toolchain is used for ML-DSA.

## ML-KEM

- Only ML-KEM-512 and ML-KEM-768 supported. ML-KEM-1024 not supported.
- Key derivation not supported.
- The data format for import and export of the key pair is the concatenation of the two 32 byte seed values: d || z

### SCE9 Usage

The SCE9 is used in Compatibility Mode for mbedCrypto acceleration. The crypto capabilities in this mode on the SCE9 are different which results in the below usage limitations with mbedCrypto:

- The module includes **both** wrapped and plaintext keys code irrespective of whether the application requires it.
- Plaintext key generation is not supported for RSA and ECC; only wrapped keys can be generated.
- If ECDH is used, only wrapped key will be generated on SCE9 and will not return an error even if the user context is somehow set for plain key. This may be relevant only if the psa_key_agreement() function with plaintext key on SCE9 is attempted.

*Note*

***For a detailed description of the different SCE9 operating modes, refer to Application Note R11AN0498.***

### Using PSA Crypto with TrustZone

Unlike FSP drivers, PSA Crypto cannot be configured as Non-secure callable in the RA Configurator for a secure project. The reason for this is that in order to achieve the security objective of controlling access to protected keys, both the PSA Crypto code as well as the keys must be placed in the secure region. Since the PSA Crypto API requires access to the keys directly during initialization and later via a key handle, allowing non-secure code to use the API by making it Non-secure callable will require the keys to be stored in non-secure memory.

This section will provide a short explanation of how to add PSA Crypto to a secure project and have it usable by the non-secure project without exposing the keys. In this example the secure project will contain an RSA private key and the non-secure project is expected to be able to perform sign and verify operations using that key.

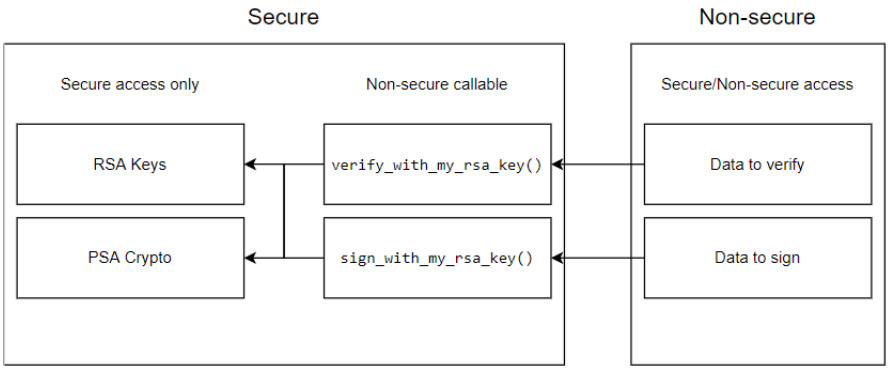



Figure 327: PSA Crypto Non-secure callable example

- Secure project
  - During secure project boot-up, psa_crypto_init() is called.
  - The RSA private key is programmed into secure flash either at the factory or by calling psa_generate_key() in persistent mode. Note that the data-flash area used by the LittleFS will have to be in the secure region if the key is generated as a persistent.
  - psa_import_key()/psa_open_key() are called with the resultant handle held in secure RAM.
  - The Non-secure callable section contains the following **user-defined** functions
  - verify_with_my_rsa_key(input_signature, input_hash, verification_result)

    - The implementation of this function in secure region will call psa_verify_hash() and return the result via verification_result.
    - sign_with_my_rsa_key(input_hash, output_signature)
    - The implementation of this function in secure region will call psa_sign_hash() and return the signature via output_signature.

- Non-secure project

    - Calls verify_with_my_rsa_key() to verify a signature. The implementation will use the public key that is present in the secure project.
    - Calls sign_with_my_rsa_key() to generate a signature. The implementation will use the private key that is present on the secure project.

  For more details on how to add user-code to the Non-secure callable region refer to the "Security Design with Arm TrustZone - IP Protection (R11AN0467EU0100)" Application Note.

# Examples

### Hash Example

This is an example on calculating the SHA256 hash using the PSA Crypto API.

```
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void psa_crypto_sha256_example (void)
{
    psa_algorithm_t      alg              = PSA_ALG_SHA_256;
    psa_hash_operation_t operation        = {0};
 size_t               expected_hash_len = PSA_HASH_LENGTH(alg);
    uint8_t             actual_hash[PSA_HASH_MAX_SIZE];
```

```
 size_t              actual_hash_len;
 mbedtls_platform_context ctx = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_setup(&operation, alg))
    {
 /* Hash setup failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_update(&operation, NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200)))
    {
 /* Hash calculation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_finish(&operation, &actual_hash[0], sizeof
(actual_hash), &actual_hash_len))
    {
 /* Reading calculated hash failed */
       debugger_break();
    }
 else if (0 != memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len))
    {
 /* Hash compare of calculated value with expected value failed */
       debugger_break();
    }
 else if (0 != memcmp(&expected_hash_len, &actual_hash_len, sizeof
(expected_hash_len)))
    {
```

```
 /* Hash size compare of calculated value with expected value failed */
       debugger_break();
    }
 else
    {
 /* SHA256 calculation succeeded */
       debugger_break();
    }
 /* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
 mbedtls_platform_teardown(&ctx);
}
```

### AES Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
static psa_status_t cipher_operation (psa_cipher_operation_t * operation,
 const uint8_t          * input,
 size_t                   input_size,
 size_t                   part_size,
                                    uint8_t               * output,
 size_t                   output_size,
 size_t                * output_len)
{
    psa_status_t status;
 size_t       bytes_to_write = 0;
 size_t       bytes_written = 0;
 size_t       len            = 0;
    *output_len = 0;
 while (bytes_written != input_size)
    {
       bytes_to_write = (input_size - bytes_written > part_size ?
                         part_size :
```

```
                        input_size - bytes_written);
      status = psa_cipher_update(operation,
                                 input + bytes_written,
                                 bytes_to_write,
                                 output + *output_len,
                                 output_size - *output_len,
                                 &len);
 if (PSA_SUCCESS != status)
     {
 return status;
     }
      bytes_written += bytes_to_write;
      *output_len   += len;
   }
   status = psa_cipher_finish(operation, output + *output_len, output_size -
*output_len, &len);
 if (PSA_SUCCESS != status)
   {
 return status;
   }
   *output_len += len;
 return status;
}
void psa_crypto_aes256cbcmultipart_example (void)
{
 enum
   {
      block_size = PSA_BLOCK_CIPHER_BLOCK_LENGTH(PSA_KEY_TYPE_AES),
      key_bits   = 256,
      input_size = 100,
      part_size = 10,
   };
 mbedtls_platform_context ctx         = {0};
 const psa_algorithm_t    alg         = PSA_ALG_CBC_PKCS7;
```

```
   psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
   psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
 size_t iv_len = 0;
   psa_key_handle_t    key_handle       = 0;
 size_t               encrypted_length = 0;
 size_t               decrypted_length = 0;
   uint8_t             iv[block_size]    = {0};
   uint8_t             input[input_size] = {0};
   uint8_t             encrypted_data[input_size + block_size] = {0};
   uint8_t             decrypted_data[input_size + block_size] = {0};
   psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
   psa_key_lifetime_t   lifetime;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
   {
 /* Platform initialization failed */
      debugger_break();
   }
 if (PSA_SUCCESS != psa_crypto_init())
   {
 /* PSA Crypto Initialization failed */
      debugger_break();
   }
 /* Set key attributes */
   psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
   psa_set_key_algorithm(&attributes, alg);
 /* To use wrapped keys instead of plaintext use PSA_KEY_TYPE_AES_WRAPPED. */
   psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
   psa_set_key_bits(&attributes, key_bits);
   lifetime = PSA_KEY_LIFETIME_VOLATILE;
 /* To use persistent keys:
  *......Use a lifetime value of PSA_KEY_LIFETIME_PERSISTENT
  * - The file system must be initialized prior to calling the generate/import key
```

```
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, input_size, part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
```

```
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
```

```
      debugger_break();
   }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
   {
 /* Destroying the key handle failed */
      debugger_break();
   }
 else
   {
 /* All the operations succeeded */
   }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
void psa_crypto_aes128xtsmultipart_example (void)
{
 enum
   {
      block_size = PSA_BLOCK_CIPHER_BLOCK_LENGTH(PSA_KEY_TYPE_AES),
      key_bits   = 256,
      input_size = 32,
      part_size = 16,
   };
 mbedtls_platform_context ctx         = {0};
 const psa_algorithm_t    alg         = PSA_ALG_XTS;
   psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
   psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
 size_t iv_len = 0;
   psa_key_handle_t    key_handle               = 0;
 size_t               encrypted_length         = 0;
 size_t               decrypted_length         = 0;
   uint8_t             iv[block_size]             = {0};
   uint8_t             input[input_size]          = {0};
   uint8_t             encrypted_data[input_size] = {0};
```

```
    uint8_t             decrypted_data[input_size] = {0};
    psa_key_attributes_t attributes                = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_lifetime_t   lifetime;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
 /* To use wrapped keys instead of plaintext use PSA_KEY_TYPE_AES_WRAPPED. */
 /* When using XTS mode, since there are two keys used, the key_bits value is twice
the AES key length.
  * For AES 128 XTS mode, the key_bits value is 256. MbedTLS only supports AES 128
currently. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
    psa_set_key_bits(&attributes, key_bits);
    lifetime = PSA_KEY_LIFETIME_VOLATILE;
 /* To use persistent keys:
  *......Use a lifetime value of PSA_KEY_LIFETIME_PERSISTENT
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
```

```
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, sizeof(input), part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
```

```
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
```

```
      debugger_break();
   }
 else
   {
 /* All the operations succeeded */
   }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
```

### AES-CCM Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding using AES-CCM.

```
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
   {
 /* Random plaintext input generation failed */
      debugger_break();
   }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
   {
 /* Key generation failed */
      debugger_break();
   }
 /* AES-CCM Encryption */
 else if (PSA_SUCCESS !=
            psa_aead_encrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                             input, sizeof(input), encrypt, sizeof(encrypt),
&output_len))
   {
 /* AES-CCM Encryption failed */
      debugger_break();
   }
```

```
/* AES-CCM Decryption */
else if (PSA_SUCCESS !=
            psa_aead_decrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                             encrypt, output_len, decrypt, sizeof(decrypt),
&output_len))
   {
/* AES-CCM Decryption failed */
      debugger_break();
   }
else if (0U != memcmp(input, decrypt, sizeof(input)))
   {
/* The decrypted result did not match the plaintext input */
      debugger_break();
   }
else
   {
/* All operations were successful */
   }
```

### AES-XTS Example

This is an example on using the PSA Crypto API to generate an AES128 XTS key, encrypting and decrypting multi-block data.

```
if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
   {
/* Random number generation for input data failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
   {
/* Generating AES 256 key and allocating to key slot failed */
      debugger_break();
   }
```

```
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, sizeof(input), part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
```

```
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded */
    }
```

### CMAC Example

This is an example on using the PSA Crypto API to generate an AES256 key, followed by generation and verification of MAC for random data of known length.

```
if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
   {
```

```
/* Random number generation failure */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
   {
/* Key generation failure */
      debugger_break();
   }
/* Steps to generate the MAC */
else if (PSA_SUCCESS != psa_mac_sign_setup(&operation, key_handle, alg))
   {
/* MAC Sign setup failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_mac_update(&operation, input, input_size))
   {
/* MAC update failed */
      debugger_break();
   }
else if (PSA_SUCCESS != psa_mac_sign_finish(&operation, AES_CMAC_mac, sizeof
(AES_CMAC_mac), &mac_ret))
   {
/* MAC Sign operation failed */
      debugger_break();
   }
else
   {
/* All the operations succeeded for MAC generation */
   }
/* Steps to verify the generated MAC */
if (PSA_SUCCESS != psa_mac_verify_setup(&verify_operation, key_handle, alg))
   {
/* MAC verification setup failure */
      debugger_break();
```

```
    }
 else if (PSA_SUCCESS != psa_mac_update(&verify_operation, input, input_size))
    {
 /* MAC update failure */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_mac_verify_finish(&verify_operation, AES_CMAC_mac,
mac_ret))
    {
 /* MAC verification failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded for MAC verification */
    }
```

### ECC Example

This is an example on using the PSA Crypto API to generate an ECC-P256R1 key, signing and verifying data after hashing it first using SHA256.

*Note*

*Unlike RSA, ECDSA does not have any padding schemes. Thus the hash argument for the ECC sign operation MUST have a size larger than or equal to the curve size; i.e. for PSA_ECC_CURVE_SECP256R1 the payload size must be at least 256/8 bytes. nist.fips.186-4: " A hash function that provides a lower security strength than the security strength associated with the bit length of 'n' ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function."*

```
#define ECC_256_BIT_LENGTH 256
#define ECC_256_EXPORTED_SIZE 500
uint8_t exportedECC_SECP256R1Key[ECC_256_EXPORTED_SIZE];
size_t exportedECC_SECP256R1Keylength = 0;
void psa_ecc256R1_example (void)
{
/* This example generates an ECC-P256R1 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
```

```
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t                   signature_length1 = 0;
 size_t                   signature_length2 = 0;
    psa_key_attributes_t    attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t    read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 mbedtls_platform_context ctx               = {0};
    psa_key_handle_t         ecc_key_handle    = {0};
    psa_hash_operation_t hash_operation = {0};
    uint8_t             payload_hash[PSA_HASH_MAX_SIZE];
 size_t               payload_hash_len;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_ECDSA(PSA_ALG_SHA_256));
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(PSA_ECC_FAMILY_SECP_R1).*/
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_FAMILY_SECP_R1));
    psa_set_key_bits(&attributes, ECC_256_BIT_LENGTH);
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
```

```
 * - Refer to the littlefs example to see how to format and mount the filesystem. */
   psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
/* Generate ECC P256R1 Key pair */
if (PSA_SUCCESS != psa_generate_key(&attributes, &ecc_key_handle))
   {
      debugger_break();
   }
/* Test the key information */
if (PSA_SUCCESS != psa_get_key_attributes(ecc_key_handle, &read_attributes))
   {
      debugger_break();
   }
/* Calculate the hash of the message */
if (PSA_SUCCESS != psa_hash_setup(&hash_operation, PSA_ALG_SHA_256))
   {
      debugger_break();
   }
if (PSA_SUCCESS != psa_hash_update(&hash_operation, payload, sizeof(payload)))
   {
      debugger_break();
   }
if (PSA_SUCCESS !=
      psa_hash_finish(&hash_operation, &payload_hash[0], sizeof(payload_hash),
&payload_hash_len))
   {
      debugger_break();
   }
/* Sign message using the private key
 * NOTE: The hash argument (payload_hash here) MUST have a size equal to the curve
size;
 * i.e. for SECP256R1 the payload size must be 256/8 bytes.
 * Similarly for SECP384R1 the payload size must be 384/8 bytes.
 * nist.fips.186-4: " A hash function that provides a lower security strength than
 * the security strength associated with the bit length of 'n' ordinarily should not
```

```
be used, since this
  * would reduce the security strength of the digital signature process to a level no
greater than that
  * provided by the hash function." */
 if (PSA_SUCCESS !=
       psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
                       signature_length1))
    {
       debugger_break();
    }
 /* Export the key. The exported key can then be save to flash for later usage. */
 if (PSA_SUCCESS !=
       psa_export_key(ecc_key_handle, exportedECC_SECP256R1Key, sizeof
(exportedECC_SECP256R1Key),
                     &exportedECC_SECP256R1Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(ecc_key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS !=
```

```
      psa_import_key(&attributes, exportedECC_SECP256R1Key,
exportedECC_SECP256R1Keylength, &ecc_key_handle))
   {
      debugger_break();
   }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
      psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
 sizeof(signature2), &signature_length2))
   {
      debugger_break();
   }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
      psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
                      signature_length2))
   {
      debugger_break();
   }
 /* Signatures cannot be compared since ECC signatures vary for the same data unless
Deterministic ECC is used which is not supported by the HW.
  * Only the verification operation can be used to validate signatures. */
}
```

### RSA Example

This is an example on using the PSA Crypto API to generate an RSA2048 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
#define RSA_2048_BIT_LENGTH 2048
#define RSA_2048_EXPORTED_SIZE 1210
/* The RSA 2048 key pair export in der format is roughly as follows
 * RSA private keys:
```

```
 * RSAPrivateKey ::= SEQUENCE { ------------------------------------ 1 + 3
 * version Version, ------------------------------------- 1 + 1 + 1
 * modulus INTEGER, ---------------- n ------------------- 1 + 3 + 256 + 1
 * publicExponent INTEGER, ---------------- e ------------------- 1 + 4
 * privateExponent INTEGER, ---------------- d ------------------- 1 + 3 + 256 (276
for Wrapped)
 * prime1 INTEGER, ---------------- p ------------------- 1 + 3 + (256 / 2)
 * prime2 INTEGER, ---------------- q ------------------- 1 + 3 + (256 / 2)
 * exponent1 INTEGER, ---------------- d mod (p-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * exponent2 INTEGER, ---------------- d mod (q-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * coefficient INTEGER, ---------------- (inverse of q) mod p - 1 + 2 + (256 / 2) (4
for Wrapped)
 * otherPrimeInfos OtherPrimeInfos OPTIONAL ----------------------- 0 (not
supported)
 * }
 */
uint8_t exportedRSA2048Key[RSA_2048_EXPORTED_SIZE];
size_t exportedRSA2048Keylength = 0;
void psa_rsa2048_example (void)
{
/* This example generates an RSA2048 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length1 = 0;
 size_t               signature_length2 = 0;
```

```
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_RSA_KEY_PAIR_WRAPPED. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate RSA 2048 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
       debugger_break();
    }
```

```
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature1,
                       signature_length1))
    {
       debugger_break();
    }
 /* Export the key */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, exportedRSA2048Key, sizeof(exportedRSA2048Key),
&exportedRSA2048Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS != psa_import_key(&attributes, exportedRSA2048Key,
exportedRSA2048Keylength, &key_handle))
    {
       debugger_break();
    }
```

```
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature2,
 sizeof(signature2), &signature_length2))
    {
       debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature2,
                       signature_length2))
    {
       debugger_break();
    }
 /* Compare signatures to verify that the same signature was generated */
 if (0 != memcmp(signature2, signature1, signature_length2))
    {
       debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
/* Generated using "openssl genrsa -out example_private_key.pem 2048" */
const uint8_t RSAKeydata[] = "-----BEGIN RSA PRIVATE KEY-----\n"
 "MIIEowIBAAKCAQEAplaPU68h4hW/eAlH2hbQl1WjoDT1znCk0O9cnE9lAnB26IUi"
 "78wUAxuTZSlrlKez9FGGIVbUMTy7dTw0ZBanogrKCaAMaRFz5LRv4I8RQzsDxxqe"
 "nud9Y6mVRWb9hyutL/yBwDd6jzAwcviAhcatTv0E5wpEJ6ezMDyaiSCbn84hfb0T"
 "ZPXQVGhmjii3f0FfFWW9ce70qkuwdkvO6HBrOGVi2++b11As+Uh/7pxyOb7wpE7K"
 "X64nohaCtXgNnV8Hr+LOPQRHEbGTCPOfx1davKEIYFvm8hVxqv20csMD5uc8AJCd"
 "0cdom13KcOpVXCnrFPNJt0lITOcH/fBzp4xoaQIDAQABAoIBAGc56qKVWeKzempk"
 "4AlRBUwVYoEGvNDLiQz8rq12fAoCf1iXXvIP5Q90qokqJlBPrtbdTO8dsnuH5BHC"
 "NgUzJf7i0TUd9PWzVgfFjnR+dMkTM6n5NB0LLf6OfEtguc+L5GOWQXNnOpDn5/lB"
```

```
 "jIj4ng8Z6FP1RAyT/xjdU03sRYfq5OPlDZ8+ck7WwQ8VxSBKsJcZVv+K4r64Vmr1"
 "MO1zfBYuwafi4oO9Z9x89Iil9GBcYuCvhLvDoWhnTUxlOTfz+Jt+qwlmVimEz6xs"
 "6ew1Ox7e6SInly3IAXe9E84yP9MDmiGkCrVr8s8GUybZ5t/yeZ3g5lPYSQ/FiuHY"
 "kvJCH4ECgYEA2IthRDkElOQGqJ9T/LjgcIVLQs0F6i7zZWg6UKv18EmG7uHd6tsZ"
 "byFfFiBat80rq6q4AUx2OOE7+LI5MxIvgRgLk14WV8H2OKK/lfxOFb6a4feGM7bH"
 "VdERQSMNeGq0bP9IOPxU1EvFp0Vfv7cfGsQhcAPgA8up61jWREhI+fECgYEAxKVT"
 "9pKinIwgjsHgmxalaNbnRf+bcqtdS98SB2MuICg8ubcBSrjm4wtGe3oCX/PbM1GV"
 "FvEDKl4TyeWFIH4MStsyDHYxwoV2C01bBHEtHTYBnjeIShMruIomsG9GeTrNmyzg"
 "dVdSg5/lxYOxuZSXvfcbyIMsW29ddgeICu9RHfkCgYBaljQibhfUkW+Xqs9fsZdy"
 "etB5KXuH9AwuJ+P9S3KfCqM/240SaoXBT5yPjQlmSpYyQkCnim0Kbm7AIw56pujo"
 "gD6Xb4y5OZLfLnYnMF0aC5qPXRTvHU9WPxeQwDEqZrkDv+der3BtPyV4TDU55klE"
 "0TeLvJNCAzkaExfPiM2+gQKBgQC5ceBYq7hGQa+CcTvLhfO/hsrbrE2AOjLllhx+"
 "cv3QvaFm0jqZqP20J7H0R/9tVZ7mKo2a8Pa3QbkPsS92kOguv7/XGK+cbhgAWJb4"
 "/XI6FfA4sM4KbUHR6hbKVGX1dYroR831WsAp+OTK+4LjLEpdj2fYFDwEjmVpJXka"
 "Ns4coQKBgGESrtpJF7OQG4xcXwR2ZiJESPvMKVmtvxRzDrc9gmoiIIIKx1fimwCv"
 "RtOz1bQcXMRw21+ZAZpen3ahWh63KC0KxMSNBJTXdczlf2uprVkSPtmAjV4qBgwv"
 "tqjmP0lnGc9wkJsVvGmMAAfQOWgxs9h/VH/b+6biEkzaaZLOyABV"
 "-----END RSA PRIVATE KEY-----\n";
/* Expected signature below is calculated using the mbedCrypto software version */
const uint8_t RSAExpectedSignature[256] =
{
    0x25, 0xC0, 0x15, 0x64, 0xD3, 0xF7, 0xC1, 0xB7, 0xA8, 0x9B, 0x56, 0x1C, 0xB5,
0xA4, 0xA5, 0x0D, 0x61, 0x52, 0x32,
    0x0C, 0x4E, 0xE8, 0xCA, 0x4B, 0x9E, 0xA2, 0x4D, 0x35, 0x0E, 0xB1, 0xA1, 0x5B, //
NOLINT(readability-magic-numbers)
    0x5B, 0xD8, 0xC1, 0x93, 0x28, 0x60, 0x90, 0x18, 0x37, 0x88, 0x66, 0xD4, 0xED,
0x37, 0x6B, 0xEC, 0x48, 0xFF, 0x3D,
    0xFB, 0x99, 0xB7, 0xEF, 0x3C, 0x8D, 0x25, 0xE9, 0xF3, 0xCF, 0x02, 0x7C, 0x7D, //
NOLINT(readability-magic-numbers)
    0xB9, 0x4D, 0x2F, 0x2F, 0x1E, 0x30, 0x48, 0x54, 0x18, 0xE4, 0x51, 0xD5, 0xC1,
0xB7, 0x3B, 0x0D, 0xE4, 0xB4, 0x19,
    0x7B, 0x9B, 0xF4, 0x35, 0x82, 0xCC, 0x91, 0x4C, 0x10, 0xE9, 0xB6, 0xF1, 0x0A, //
NOLINT(readability-magic-numbers)
    0x23, 0xEB, 0x7D, 0x51, 0x47, 0x36, 0xFE, 0x13, 0xAF, 0x3C, 0x23, 0x9F, 0x7E,
```

```
0xFC, 0xCF, 0x7A, 0x7C, 0x2D, 0xDB,
    0xD9, 0xDA, 0xEB, 0xF7, 0xB5, 0x6B, 0x8D, 0xE0, 0x18, 0x9A, 0x5B, 0xB7, 0x0A, //
NOLINT(readability-magic-numbers)
    0xA3, 0x4E, 0xE1, 0xB7, 0xF7, 0xD1, 0x94, 0xD5, 0x7A, 0xD3, 0x27, 0xE2, 0x1F,
0x3A, 0xEB, 0xF0, 0x83, 0x10, 0x52,
    0x51, 0x5F, 0x58, 0xF8, 0x81, 0x42, 0x48, 0x83, 0x2D, 0xF0, 0xA9, 0x7D, 0x79, //
NOLINT(readability-magic-numbers)
    0x2B, 0xF1, 0x68, 0xC2, 0x22, 0xC0, 0x0C, 0x72, 0x63, 0x37, 0xBF, 0xEC, 0x72,
0x97, 0xD4, 0xA5, 0x91, 0x2E, 0x1F,
    0xA3, 0x78, 0x9A, 0xCE, 0xFE, 0x27, 0x7F, 0x2B, 0x85, 0x7D, 0x22, 0x2C, 0x0D, //
NOLINT(readability-magic-numbers)
    0x1E, 0x10, 0xB7, 0xFF, 0x9A, 0xA7, 0x99, 0xD2, 0xB9, 0x40, 0x53, 0xB3, 0xA9,
0x52, 0x5D, 0xBD, 0xC8, 0x12, 0x8D,
    0x39, 0xD7, 0x97, 0x03, 0xD2, 0x80, 0x21, 0xC3, 0xA7, 0x8B, 0xE3, 0x3D, 0xF0, //
NOLINT(readability-magic-numbers)
    0x4D, 0x4C, 0x4D, 0xC4, 0xC7, 0xE5, 0xE4, 0x35, 0x75, 0xAA, 0x45, 0x3B, 0x9C,
0x64, 0xC1, 0x94, 0x6E, 0x15, 0x0A,
    0xE8, 0x84, 0xCD, 0xFC, 0x7A, 0xBC, 0x5C, 0x8C, 0xA8, 0x95, 0x07, 0x79, 0x4E, //
NOLINT(readability-magic-numbers)
};
void psa_rsa2048_pem_format_import_example (void)
{
/* This example imports an RSA2048 keypair of PEM format, performs signing and
verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length = 0U;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
```

```
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* Import the private key (This includes both the private and public key) */
    mbedtls_pk_context ctx_rsa;
    mbedtls_pk_init(&ctx_rsa);
 /* For keys other than ASN format, mbedtls_pk_parse_key and
mbedtls_pk_import_into_psa APIs must be called instead of psa_import_key */
 if (PSA_SUCCESS !=
       mbedtls_pk_parse_key(&ctx_rsa, RSAKeydata, sizeof(RSAKeydata), NULL, 0,
mbedtls_psa_get_random,
                            MBEDTLS_PSA_RANDOM_STATE))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != mbedtls_pk_import_into_psa(&ctx_rsa, &attributes, &key_handle))
    {
       debugger_break();
    }
    mbedtls_pk_free(&ctx_rsa);
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Sign message using the key */
```

```
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature, sizeof(signature),
                     &signature_length))
    {
       debugger_break();
    }
 if (0 != memcmp(signature, &RSAExpectedSignature, sizeof(RSAExpectedSignature)))
    {
       debugger_break();
    }
 /* Verify the signature */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), RSAExpectedSignature,
 sizeof(RSAExpectedSignature)))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
}
```

### CHACHA20 Example

This is an example on using the PSA Crypto API to import an ChaCha20 key, encrypting and decrypting single block data.

```
/* Test vector from http://www.rfc-editor.org/rfc/rfc8439.html#section-2.8.2 */
const uint8_t chacha_key[] =
```

```
{
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, // NOLINT(readability-magic-
numbers)
    0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f, // NOLINT(readability-magic-
numbers)
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, // NOLINT(readability-magic-
numbers)
    0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f // NOLINT(readability-magic-
numbers)
};
const uint8_t chacha_iv[] =
{
    0x07, 0x00, 0x00, 0x00,                       /* 32-bit common part */
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47 /* 64-bit IV */
};
const uint8_t chacha_plaintext[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C,
0x0D, 0x0E, 0x0F, // NOLINT(readability-magic-numbers)
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
0x1D, 0x1E, 0x1F, // NOLINT(readability-magic-numbers)
    0x20, 0x21, 0x22,
0x23                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chacha_ciphertext[] =
{
    0x9F, 0x7A, 0xEB, 0x5E, 0x05, 0xF8, 0x46, 0xBD, 0x1D, 0xEB, 0x85, 0xF0, 0x3A,
0x8C, 0x04, 0xA1, // NOLINT(readability-magic-numbers)
    0xD1, 0xD1, 0x9A, 0x2C, 0x1D, 0x14, 0x78, 0xC9, 0xC5, 0x93, 0xCA, 0x9C, 0x49,
0x9F, 0x1D, 0xBA, // NOLINT(readability-magic-numbers)
    0x6E, 0xBF, 0xE9,
0x1B                                                                    //
NOLINT(readability-magic-numbers)
```

```
};
const uint8_t chacha_mac[] =
{
    0xC8, 0xB3, 0xFA, 0xDC, 0xF9, 0x1F, 0x78, 0x97, // NOLINT(readability-magic-
numbers)
    0xCD, 0xD7, 0x54, 0xE2, 0xFC, 0x0B, 0xE1, 0xE5 // NOLINT(readability-magic-
numbers)
};
typedef struct import_key_items_symmetric_s
{
 const uint8_t * key;
 size_t          key_size;
 const uint8_t * nonce_iv;          // Nomenclature 'nonce' OR 'iv' depends on the
algorithm. Eg: This is Nonce for CCM and IV for GCM.
 size_t          nonce_iv_size;
 const uint8_t * aad;
 size_t          aad_size;
 const uint8_t * mac;
 size_t          mac_size;
 const uint8_t * plaintext;
 size_t          plaintext_size;
 const uint8_t * ciphertext;
 size_t          ciphertext_size;
} import_key_items_symmetric_t;
const import_key_items_symmetric_t chacha_testdata =
{
    chacha_key,
 sizeof(chacha_key),
    chacha_iv,
 sizeof(chacha_iv),
    NULL,
    0,
    chacha_mac,
    16,
```

```
    chacha_plaintext,
 sizeof(chacha_plaintext),
    chacha_ciphertext,
 sizeof(chacha_ciphertext),
};
void psa_crypto_chacha20_example (void)
{
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = 0;
 size_t encr_len = 0;
 size_t decr_len = 0;
    uint8_t encrypt[200];             // NOLINT(readability-magic-numbers)
    uint8_t decrypt[200];             // NOLINT(readability-magic-numbers)
    import_key_items_symmetric_t * p_import_key_items = (import_key_items_symmetric_t
*) &chacha_testdata;
    psa_key_attributes_t          attributes         = PSA_KEY_ATTRIBUTES_INIT;
 const psa_algorithm_t          alg                = PSA_ALG_STREAM_CIPHER;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_CHACHA20);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
```

```
    psa_set_key_bits(&attributes, 256); // NOLINT(readability-magic-numbers)
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(PSA_KEY_LIFETIME_VOLATILE))
    {
       psa_set_key_id(&attributes, (psa_key_id_t) 2);
    }
 if (PSA_SUCCESS != psa_import_key(&attributes, p_import_key_items->key,
p_import_key_items->key_size, &key_handle))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_cipher_encrypt(key_handle, alg, p_import_key_items->plaintext,
p_import_key_items->plaintext_size, encrypt,
 sizeof(encrypt), &encr_len))
    {
 /* Encryption failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_cipher_decrypt(key_handle, alg, encrypt, encr_len, decrypt,
sizeof(decrypt), &decr_len))
    {
 /* Decryption failed */
       debugger_break();
    }
 if (p_import_key_items->plaintext_size == decr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->plaintext, decrypt,
p_import_key_items->plaintext_size))
    {
       debugger_break();
    }
 /* Destroy the key */
```

```
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
       debugger_break();
    }
}
```

### CHACHA20-POLY1305 Example

This is an example on using the PSA Crypto API to import an ChaCha20 key, encrypting/decrypting single block data and generates the MAC (authentication tag)

```
const uint8_t chachapoly_key[] =
{
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, // NOLINT(readability-magic-
numbers)
    0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f, // NOLINT(readability-magic-
numbers)
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, // NOLINT(readability-magic-
numbers)
    0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f // NOLINT(readability-magic-
numbers)
};
const uint8_t chachapoly_iv[] =
{
    0x07, 0x00, 0x00, 0x00,                       /* 32-bit common part */
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47 /* 64-bit IV */
};
const uint8_t chachapoly_plaintext[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C,
0x0D, 0x0E, 0x0F, // NOLINT(readability-magic-numbers)
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
0x1D, 0x1E, 0x1F, // NOLINT(readability-magic-numbers)
    0x20, 0x21, 0x22,
```

```
0x23                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chachapoly_ciphertext[] =
{
    0x9F, 0x7A, 0xEB, 0x5E, 0x05, 0xF8, 0x46, 0xBD, 0x1D, 0xEB, 0x85, 0xF0, 0x3A,
0x8C, 0x04, 0xA1, // NOLINT(readability-magic-numbers)
    0xD1, 0xD1, 0x9A, 0x2C, 0x1D, 0x14, 0x78, 0xC9, 0xC5, 0x93, 0xCA, 0x9C, 0x49,
0x9F, 0x1D, 0xBA, // NOLINT(readability-magic-numbers)
    0x6E, 0xBF, 0xE9,
0x1B                                                                    //
NOLINT(readability-magic-numbers)
};
const uint8_t chachapoly_mac[] =
{
    0xC8, 0xB3, 0xFA, 0xDC, 0xF9, 0x1F, 0x78, 0x97, // NOLINT(readability-magic-
numbers)
    0xCD, 0xD7, 0x54, 0xE2, 0xFC, 0x0B, 0xE1, 0xE5 // NOLINT(readability-magic-
numbers)
};
// typedef struct import_key_items_symmetric_s
// {
// const uint8_t * key;
// size_t key_size;
// const uint8_t * nonce_iv; // Nomenclature 'nonce' OR 'iv' depends on the
algorithm. Eg: This is Nonce for CCM and IV for GCM.
// size_t nonce_iv_size;
// const uint8_t * aad;
// size_t aad_size;
// const uint8_t * mac;
// size_t mac_size;
// const uint8_t * plaintext;
// size_t plaintext_size;
// const uint8_t * ciphertext;
```

```
// size_t ciphertext_size;
// } import_key_items_symmetric_t;
const import_key_items_symmetric_t chachapoly_testdata =
{
    chachapoly_key,
 sizeof(chachapoly_key),
    chachapoly_iv,
 sizeof(chachapoly_iv),
    NULL,
    0,
    chachapoly_mac,
    16,
    chachapoly_plaintext,
 sizeof(chachapoly_plaintext),
    chachapoly_ciphertext,
 sizeof(chachapoly_ciphertext),
};
void psa_crypto_chachapoly_example (void)
{
 enum
    {
       block_size = 16,
       part_size = block_size,
    };
 const psa_algorithm_t    alg       = PSA_ALG_CHACHA20_POLY1305;
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = 0;
 size_t encr_len = 0;
 size_t decr_len = 0;
    uint8_t encrypt[200];             // NOLINT(readability-magic-numbers)
    uint8_t decrypt[200];             // NOLINT(readability-magic-numbers)
    import_key_items_symmetric_t * p_import_key_items = (import_key_items_symmetric_t
*) &chachapoly_testdata;
    psa_key_attributes_t          attributes         = PSA_KEY_ATTRIBUTES_INIT;
```

```
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_CHACHA20);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
    psa_set_key_bits(&attributes, 256); // NOLINT(readability-magic-numbers)
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(PSA_KEY_LIFETIME_VOLATILE))
    {
       psa_set_key_id(&attributes, (psa_key_id_t) 2);
    }
 if (PSA_SUCCESS != psa_import_key(&attributes, p_import_key_items->key,
p_import_key_items->key_size, &key_handle))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_aead_encrypt(key_handle, alg, p_import_key_items->nonce_iv,
p_import_key_items->nonce_iv_size, NULL, 0,
                        p_import_key_items->plaintext,
p_import_key_items->plaintext_size, encrypt, sizeof(encrypt),
                        &encr_len))
    {
```

```
       debugger_break();
    }
 if (p_import_key_items->plaintext_size + p_import_key_items->mac_size != encr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->ciphertext, encrypt,
p_import_key_items->plaintext_size))
    {
       debugger_break();
    }
 if (0 !=
       memcmp(p_import_key_items->mac, &encrypt[p_import_key_items->plaintext_size],
p_import_key_items->mac_size))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_aead_decrypt(key_handle, alg, p_import_key_items->nonce_iv,
p_import_key_items->nonce_iv_size, NULL, 0,
                        encrypt, encr_len, decrypt, sizeof(decrypt), &decr_len))
    {
       debugger_break();
    }
 if (p_import_key_items->plaintext_size != decr_len)
    {
       debugger_break();
    }
 if (0 != memcmp(p_import_key_items->plaintext, decrypt,
p_import_key_items->plaintext_size))
    {
       debugger_break();
    }
 /* Destroy the key */
```

```
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
}
```

### ML-KEM Example

This is an example on using the PSA Crypto API to generate an MLKEM-512 key, encapsulating and decapsulating ciphertext.

```
void psa_mlkem512_keygen_example (void)
{
 mbedtls_platform_context ctx                  = {0};
 size_t               bits                     = PSA_KEY_BITS_ML_KEM_512;
    psa_key_handle_t    key_handle               = 0;
    psa_key_id_t         output_key_handle       = 0;
    psa_key_id_t         decap_output_key_handle = 0;
    psa_key_attributes_t attributes               = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t shared_secret_attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t
ciphertext[PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512)];
 size_t               ciphertext_len =
PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512);
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
```

```
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
    psa_set_key_algorithm(&attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_KEM_KEY_PAIR);
    psa_set_key_bits(&attributes, bits);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&shared_secret_attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
    psa_set_key_algorithm(&shared_secret_attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&shared_secret_attributes, PSA_KEY_TYPE_RAW_DATA);
    psa_set_key_bits(&shared_secret_attributes,
PSA_BYTES_TO_BITS(PSA_ML_KEM_SHARED_SECRET_SIZE));
    psa_set_key_lifetime(&shared_secret_attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS !=
       psa_encapsulate(key_handle, PSA_ALG_ML_KEM, &shared_secret_attributes,
&output_key_handle, &ciphertext[0],
                       ciphertext_len, &ciphertext_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_decapsulate(key_handle, PSA_ALG_ML_KEM, &ciphertext[0], ciphertext_len,
&shared_secret_attributes,
                       &decap_output_key_handle))
    {
 /* Key decapsulation failed */
```

```
       debugger_break();
    }
}
/* Two 32-byte seed values, d and z, concatenated */
uint8_t mlkem_import_keypair[PSA_KEY_EXPORT_ML_KEM_KEY_PAIR_MAX_SIZE] =
{
    0x1E, 0xB4, 0x40, 0x0A, 0x01, 0x62, 0x9D, 0x51, 0x79, 0x74, 0xE2, 0xCD, 0x85,
0xB9, 0xDE, 0xF5, // NOLINT(readability-magic-numbers)
    0x90, 0x82, 0xDE, 0x50, 0x8E, 0x6F, 0x9C, 0x2B, 0x0E, 0x34, 0x1E, 0x12, 0x96,
0x59, 0x55, 0xCA, // NOLINT(readability-magic-numbers)
    0x1A, 0x39, 0x41, 0x11, 0x16, 0x38, 0x03, 0xFE, 0x2E, 0x85, 0x19, 0xC3, 0x35,
0xA6, 0x86, 0x75, // NOLINT(readability-magic-numbers)
    0x56, 0x33, 0x8E, 0xAD, 0xAF, 0xA2, 0x2B, 0x5F, 0xC5, 0x57, 0x43, 0x05, 0x60,
0xCC, 0xD6, 0x93 // NOLINT(readability-magic-numbers)
};
void psa_mlkem512_keypair_import_export_example (void)
{
 mbedtls_platform_context ctx                  = {0};
 size_t               bits                     = PSA_KEY_BITS_ML_KEM_512;
    psa_key_handle_t    key_handle               = 0;
    psa_key_id_t         output_key_handle       = 0;
    psa_key_id_t         decap_output_key_handle = 0;
    psa_key_attributes_t attributes               = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t shared_secret_attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t
ciphertext[PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512)];
 size_t               ciphertext_len =
PSA_ML_KEM_CIPHERTEXT_SIZE(PSA_KEY_BITS_ML_KEM_512);
    uint8_t
export_keypair[PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(PSA_KEY_BITS_ML_KEM_512)];
 size_t               export_keypair_len = 0;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
```

```
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_KEM_KEY_PAIR);
    psa_set_key_bits(&attributes, bits);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Import key pair */
 if (PSA_SUCCESS !=
       psa_import_key(&attributes, &mlkem_import_keypair[0],
PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(bits), &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&shared_secret_attributes, PSA_KEY_USAGE_ENCAPSULATE |
PSA_KEY_USAGE_DECAPSULATE);
    psa_set_key_algorithm(&shared_secret_attributes, PSA_ALG_ML_KEM);
    psa_set_key_type(&shared_secret_attributes, PSA_KEY_TYPE_RAW_DATA);
    psa_set_key_bits(&shared_secret_attributes,
PSA_BYTES_TO_BITS(PSA_ML_KEM_SHARED_SECRET_SIZE));
    psa_set_key_lifetime(&shared_secret_attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS !=
       psa_encapsulate(key_handle, PSA_ALG_ML_KEM, &shared_secret_attributes,
&output_key_handle, &ciphertext[0],
```

```
                        ciphertext_len, &ciphertext_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_decapsulate(key_handle, PSA_ALG_ML_KEM, &ciphertext[0], ciphertext_len,
&shared_secret_attributes,
                        &decap_output_key_handle))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
 /* Export the key pair */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, &export_keypair[0],
PSA_KEY_GEN_ML_KEM_KEY_PAIR_MAX_SIZE(bits), &export_keypair_len))
    {
 /* Key pair export failed */
       debugger_break();
    }
 /* Make sure the exported key pair matches the imported buffer */
 if (0 != memcmp(mlkem_import_keypair, export_keypair, export_keypair_len))
    {
 /* Exported key does not match imported key */
       debugger_break();
    }
}
```

### ML-DSA Example

This is an example on using the PSA Crypto API to generate an MLDSA-44 key, sign and verify a message.

```
#define MLDSA_MESSAGE_LEN 512U
```

```
static uint8_t mldsa_test_message[MLDSA_MESSAGE_LEN] =
{
    0xF0, 0xA4, 0xA1, 0x8C, 0xCF, 0x98, 0x2E, 0x89, 0x1F, 0x3E, 0x91, 0x73, 0x27,
0x90, 0x1F, 0x73, // NOLINT(readability-magic-numbers)
    0x83, 0x90, 0xE1, 0x7A, 0xB9, 0x5F, 0xF1, 0xFB, 0x7C, 0x76, 0x83, 0x50, 0x38,
0x88, 0x9F, 0x73, // NOLINT(readability-magic-numbers)
    0xAF, 0x0D, 0x24, 0x49, 0x76, 0xDB, 0x62, 0x21, 0x94, 0xFA, 0xDA, 0x41, 0x14,
0x9E, 0x53, 0x4A, // NOLINT(readability-magic-numbers)
    0xCD, 0x96, 0xB9, 0x23, 0x38, 0x52, 0x98, 0x14, 0xEE, 0x18, 0x68, 0x32, 0xB9,
0xC3, 0x83, 0xF9, // NOLINT(readability-magic-numbers)
    0x80, 0xA5, 0x11, 0x7B, 0xDB, 0xED, 0x09, 0x3D, 0xB3, 0x1A, 0xCD, 0x52, 0x73,
0xDF, 0x13, 0xE5, // NOLINT(readability-magic-numbers)
    0xA8, 0x3D, 0x14, 0xBB, 0x82, 0x1F, 0x18, 0xC1, 0x88, 0xFF, 0x36, 0x80, 0x19,
0x16, 0x3C, 0x61, // NOLINT(readability-magic-numbers)
    0x4C, 0x4D, 0x20, 0x27, 0x83, 0xFC, 0x88, 0xE2, 0x52, 0x3C, 0x42, 0x51, 0x14,
0x21, 0x14, 0x25, // NOLINT(readability-magic-numbers)
    0x98, 0x81, 0x48, 0xF5, 0x25, 0xD0, 0x47, 0xA0, 0xCD, 0x95, 0x77, 0x63, 0xCF,
0x5B, 0x79, 0xF6, // NOLINT(readability-magic-numbers)
    0x6B, 0xE1, 0x73, 0x87, 0x04, 0xAC, 0x66, 0xF1, 0x33, 0x39, 0x8B, 0xC9, 0xC0,
0xE5, 0x57, 0x00, // NOLINT(readability-magic-numbers)
    0x63, 0x06, 0xDB, 0x92, 0x0C, 0xA8, 0x43, 0xC0, 0xD5, 0x24, 0xD4, 0xEA, 0xED,
0xBC, 0x1B, 0xDE, // NOLINT(readability-magic-numbers)
    0x05, 0x03, 0x37, 0x6B, 0x22, 0x77, 0x2B, 0x74, 0x14, 0x55, 0x9D, 0x68, 0x90,
0x1C, 0xC5, 0xA1, // NOLINT(readability-magic-numbers)
    0x80, 0x65, 0x20, 0xEB, 0x39, 0x7F, 0xFA, 0xA2, 0x4D, 0x96, 0xFF, 0x2F, 0x20,
0x7B, 0x47, 0xD1, // NOLINT(readability-magic-numbers)
    0xA2, 0x1C, 0xE6, 0x86, 0x87, 0xD7, 0x65, 0x83, 0xF2, 0x15, 0x11, 0x07, 0x4B,
0x90, 0x56, 0xD4, // NOLINT(readability-magic-numbers)
    0x54, 0xF5, 0x65, 0x35, 0x8D, 0xBE, 0xFA, 0xEC, 0x8A, 0xBE, 0x43, 0x88, 0xFB,
0x1D, 0x8A, 0x25, // NOLINT(readability-magic-numbers)
    0x5D, 0xE8, 0xF9, 0xE7, 0xF1, 0x1A, 0x96, 0xE6, 0x9A, 0xAD, 0xA9, 0x34, 0x6C,
0x84, 0x61, 0x90, // NOLINT(readability-magic-numbers)
    0x1E, 0xFA, 0x4D, 0xAA, 0xA1, 0xA4, 0xBA, 0x9B, 0x04, 0xD8, 0xA1, 0x1E, 0xBD,
0x00, 0x31, 0x9B, // NOLINT(readability-magic-numbers)
```

```
    0x4A, 0x7D, 0x8B, 0xAC, 0x21, 0x74, 0xCF, 0xA2, 0xA0, 0x06, 0xD9, 0x48, 0x59,
0xFE, 0xC5, 0x2B, // NOLINT(readability-magic-numbers)
    0xDC, 0x01, 0x88, 0x2C, 0xB9, 0xD4, 0xA3, 0xE8, 0x95, 0x1A, 0x5C, 0xC1, 0xBD,
0x36, 0xEC, 0x74, // NOLINT(readability-magic-numbers)
    0xF6, 0x03, 0x33, 0x31, 0xB0, 0x3E, 0x92, 0xA7, 0x27, 0x07, 0x7F, 0x44, 0x8D,
0x4F, 0xB5, 0x60, // NOLINT(readability-magic-numbers)
    0x9A, 0x5E, 0x55, 0xAA, 0x75, 0x28, 0x9F, 0x7D, 0xAF, 0x3B, 0xEA, 0xB3, 0x1C,
0xF7, 0xAA, 0x1F, // NOLINT(readability-magic-numbers)
    0x4B, 0x66, 0x98, 0x0F, 0x5F, 0x5F, 0x8A, 0x31, 0xAC, 0xFE, 0x55, 0x58, 0x52,
0x52, 0x24, 0x7F, // NOLINT(readability-magic-numbers)
    0xCD, 0x78, 0xB9, 0x67, 0x56, 0x38, 0xDB, 0x8E, 0x12, 0x92, 0x91, 0x3A, 0x6F,
0x3E, 0xA5, 0x7C, // NOLINT(readability-magic-numbers)
    0x60, 0xD1, 0x97, 0xB1, 0xCF, 0x83, 0xD8, 0x53, 0xC6, 0xAB, 0xE3, 0x50, 0xB7,
0xB1, 0x10, 0x40, // NOLINT(readability-magic-numbers)
    0x76, 0x5A, 0xF1, 0xC1, 0x74, 0x0A, 0xF1, 0xE3, 0x18, 0x68, 0x92, 0x37, 0xB3,
0x50, 0xDD, 0x8F, // NOLINT(readability-magic-numbers)
    0x4F, 0x0C, 0x0A, 0x63, 0x93, 0x9A, 0xDA, 0xBF, 0x39, 0x2D, 0x71, 0x47, 0x1B,
0xF3, 0xF7, 0xA5, // NOLINT(readability-magic-numbers)
    0xCA, 0x44, 0x8A, 0x4D, 0xCD, 0xF4, 0x89, 0x6A, 0x34, 0x2B, 0x4B, 0xA9, 0x42,
0x12, 0x81, 0x37, // NOLINT(readability-magic-numbers)
    0xF3, 0xFF, 0x44, 0xAE, 0x57, 0xDB, 0x85, 0x00, 0xA7, 0xBF, 0xB6, 0xFD, 0x74,
0x25, 0x0E, 0x63, // NOLINT(readability-magic-numbers)
    0x12, 0xD5, 0x4A, 0x35, 0x88, 0xD0, 0x8E, 0x93, 0xBA, 0x62, 0x3E, 0x18, 0x9B,
0x81, 0x7E, 0xEF, // NOLINT(readability-magic-numbers)
    0x2B, 0xE6, 0x6D, 0x2A, 0x57, 0x09, 0x7B, 0x14, 0x7A, 0x13, 0x1D, 0x5C, 0x88,
0x43, 0xCC, 0x73, // NOLINT(readability-magic-numbers)
    0x43, 0xE6, 0x2B, 0xC0, 0x02, 0xAD, 0x1E, 0x60, 0x29, 0x9B, 0xB7, 0xD2, 0xF0,
0xFF, 0x14, 0xC5, // NOLINT(readability-magic-numbers)
    0x1B, 0x3E, 0x41, 0x41, 0x98, 0x52, 0x4A, 0x3D, 0x96, 0x4C, 0xE2, 0xA6, 0x31,
0xB7, 0x2F, 0x71, // NOLINT(readability-magic-numbers)
    0x34, 0x2E, 0x17, 0xAF, 0x47, 0x99, 0x2E, 0x6C, 0x27, 0x06, 0x7C, 0xA1, 0x47,
0x11, 0x61, 0x5C // NOLINT(readability-magic-numbers)
};
void psa_mkdsa44_example (void)
```

```
{
 mbedtls_platform_context ctx       = {0};
    psa_algorithm_t          alg       = PSA_ALG_HASH_ML_DSA(PSA_ALG_SHAKE256);
    psa_key_handle_t         key_handle = 0;
    psa_key_attributes_t    attributes = PSA_KEY_ATTRIBUTES_INIT;
 size_t sign_len;
    uint8_t sign[PSA_ML_DSA_SIGNATURE_SIZE(PSA_KEY_BITS_ML_DSA_44)];
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes,
                            PSA_KEY_USAGE_SIGN_MESSAGE | PSA_KEY_USAGE_VERIFY_MESSAGE
| PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_HASH_ML_DSA(PSA_ALG_SHAKE256));
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_KEY_PAIR);
    psa_set_key_bits(&attributes, PSA_KEY_BITS_ML_DSA_44);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_sign_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sizeof(sign),
```

```
                        &sign_len))
    {
 /* Key encapsulation failed */
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
    {
 /* Key decapsulation failed */
       debugger_break();
    }
}
/* The 32-byte seed value */
uint8_t mldsa_import_keypair[PSA_KEY_EXPORT_ML_DSA_KEY_PAIR_MAX_SIZE] =
{
    0x4B, 0xE7, 0xA0, 0x1A, 0x99, 0xA5, 0xE5, 0xBC, 0xFE, 0x3C, 0x06, 0x78, 0x5D,
0x8E, 0x4E, 0xC6, // NOLINT(readability-magic-numbers)
    0x64, 0x08, 0x22, 0x27, 0xD8, 0x67, 0x04, 0xE9, 0xE4, 0x48, 0x62, 0x62, 0x3A,
0x05, 0xC8, 0xB3, // NOLINT(readability-magic-numbers)
};
void psa_mldsa44_keypair_import_export_example (void)
{
    psa_key_id_t         key_handle = 0;
    psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
    uint8_t             sign[PSA_ML_DSA_SIGNATURE_SIZE(PSA_KEY_BITS_ML_DSA_44)]
__ALIGNED(4);
 size_t              sign_len;
    uint8_t
public_key[PSA_KEY_GEN_ML_DSA_PUB_KEY_SIZE(PSA_KEY_BITS_ML_DSA_44)];
 size_t              public_key_len = 0;
    psa_algorithm_t      alg           = PSA_ALG_HASH_ML_DSA(PSA_ALG_SHAKE256);
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_MESSAGE |
```

```
PSA_KEY_USAGE_VERIFY_MESSAGE);
    psa_set_key_algorithm(&attributes, alg);
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_KEY_PAIR);
    psa_set_key_bits(&attributes, PSA_KEY_BITS_ML_DSA_44);
 /* Import key pair */
 if (PSA_SUCCESS !=
       psa_import_key(&attributes, &mldsa_import_keypair[0],
                      PSA_KEY_GEN_ML_DSA_KEY_PAIR_MAX_SIZE(PSA_KEY_BITS_ML_DSA_44),
&key_handle))
    {
 /* Key import failed */
       debugger_break();
    }
 /* Sign the message */
 if (PSA_SUCCESS !=
       psa_sign_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sizeof(sign),
                        &sign_len))
    {
 /* Sign message failed */
       debugger_break();
    }
 /* Verify the signature */
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
    {
 /* Verify message failed */
       debugger_break();
    }
 /* Export public key */
 if (PSA_SUCCESS !=
       psa_export_public_key(key_handle, &public_key[0],
```

```
PSA_KEY_GEN_ML_DSA_PUB_KEY_SIZE(PSA_KEY_BITS_ML_DSA_44),
                                &public_key_len))
    {
 /* Public key export failed */
       debugger_break();
    }
 /* Destroy key pair */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Key destroy failed */
       debugger_break();
    }
 /* Import public key */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ML_DSA_PUBLIC_KEY);
 if (PSA_SUCCESS != psa_import_key(&attributes, &public_key[0], public_key_len,
&key_handle))
    {
 /* Key import failed */
       debugger_break();
    }
 /* Verify signature */
 if (PSA_SUCCESS !=
       psa_verify_message(key_handle, alg, mldsa_test_message, sizeof
(mldsa_test_message), sign, sign_len))
    {
 /* Verify message failed */
       debugger_break();
    }
}
```

### Migrating to MbedTLS 3.6.0

MBEDTLS_PSA_ASSUME_EXCLUSIVE_BUFFERS macro is disabled by default to handle shared memory in a secure manner. Note that this increases heap memory usage and code size. The macro can be enabled if all buffers passed to any PSA function reside in memory that is accessible only to the PSA function during its execution or if it possible otherwise for the user to conclude that the user buffers

passed into PSA are secure from modification. Refer to https://github.com/Mbed-TLS/mbedtls-docs/blob/main/security-advisories/mbedtls-security-advisory-2024-03.md for more details.

psa_import_key() now only accepts RSA keys in the PSA standard formats. Applications that need other formats like: PKCS#8, SubjectPublicKey, PEM can call mbedtls_pk_parse_{public,}key() followed by mbedtls_pk_import_into_psa().

```
/* Generated using "openssl genrsa -out example_private_key.pem 2048" */
const uint8_t RSAKeydata[] = "-----BEGIN RSA PRIVATE KEY-----\n"
 "MIIEowIBAAKCAQEAplaPU68h4hW/eAlH2hbQl1WjoDT1znCk0O9cnE9lAnB26IUi"
 "78wUAxuTZSlrlKez9FGGIVbUMTy7dTw0ZBanogrKCaAMaRFz5LRv4I8RQzsDxxqe"
 "nud9Y6mVRWb9hyutL/yBwDd6jzAwcviAhcatTv0E5wpEJ6ezMDyaiSCbn84hfb0T"
 "ZPXQVGhmjii3f0FfFWW9ce70qkuwdkvO6HBrOGVi2++b11As+Uh/7pxyOb7wpE7K"
 "X64nohaCtXgNnV8Hr+LOPQRHEbGTCPOfx1davKEIYFvm8hVxqv20csMD5uc8AJCd"
 "0cdom13KcOpVXCnrFPNJt0lITOcH/fBzp4xoaQIDAQABAoIBAGc56qKVWeKzempk"
 "4AlRBUwVYoEGvNDLiQz8rq12fAoCf1iXXvIP5Q90qokqJlBPrtbdTO8dsnuH5BHC"
 "NgUzJf7i0TUd9PWzVgfFjnR+dMkTM6n5NB0LLf6OfEtguc+L5GOWQXNnOpDn5/lB"
 "jIj4ng8Z6FP1RAyT/xjdU03sRYfq5OPlDZ8+ck7WwQ8VxSBKsJcZVv+K4r64Vmr1"
 "MO1zfBYuwafi4oO9Z9x89Iil9GBcYuCvhLvDoWhnTUxlOTfz+Jt+qwlmVimEz6xs"
 "6ew1Ox7e6SInly3IAXe9E84yP9MDmiGkCrVr8s8GUybZ5t/yeZ3g5lPYSQ/FiuHY"
 "kvJCH4ECgYEA2IthRDkElOQGqJ9T/LjgcIVLQs0F6i7zZWg6UKv18EmG7uHd6tsZ"
 "byFfFiBat80rq6q4AUx2OOE7+LI5MxIvgRgLk14WV8H2OKK/lfxOFb6a4feGM7bH"
 "VdERQSMNeGq0bP9IOPxU1EvFp0Vfv7cfGsQhcAPgA8up61jWREhI+fECgYEAxKVT"
 "9pKinIwgjsHgmxalaNbnRf+bcqtdS98SB2MuICg8ubcBSrjm4wtGe3oCX/PbM1GV"
 "FvEDKl4TyeWFIH4MStsyDHYxwoV2C01bBHEtHTYBnjeIShMruIomsG9GeTrNmyzg"
 "dVdSg5/lxYOxuZSXvfcbyIMsW29ddgeICu9RHfkCgYBaljQibhfUkW+Xqs9fsZdy"
 "etB5KXuH9AwuJ+P9S3KfCqM/240SaoXBT5yPjQlmSpYyQkCnim0Kbm7AIw56pujo"
 "gD6Xb4y5OZLfLnYnMF0aC5qPXRTvHU9WPxeQwDEqZrkDv+der3BtPyV4TDU55klE"
 "0TeLvJNCAzkaExfPiM2+gQKBgQC5ceBYq7hGQa+CcTvLhfO/hsrbrE2AOjLllhx+"
 "cv3QvaFm0jqZqP20J7H0R/9tVZ7mKo2a8Pa3QbkPsS92kOguv7/XGK+cbhgAWJb4"
 "/XI6FfA4sM4KbUHR6hbKVGX1dYroR831WsAp+OTK+4LjLEpdj2fYFDwEjmVpJXka"
 "Ns4coQKBgGESrtpJF7OQG4xcXwR2ZiJESPvMKVmtvxRzDrc9gmoiIIIKx1fimwCv"
 "RtOz1bQcXMRw21+ZAZpen3ahWh63KC0KxMSNBJTXdczlf2uprVkSPtmAjV4qBgwv"
 "tqjmP0lnGc9wkJsVvGmMAAfQOWgxs9h/VH/b+6biEkzaaZLOyABV"
 "-----END RSA PRIVATE KEY-----\n";
/* Expected signature below is calculated using the mbedCrypto software version */
const uint8_t RSAExpectedSignature[256] =
```

```
{
    0x25, 0xC0, 0x15, 0x64, 0xD3, 0xF7, 0xC1, 0xB7, 0xA8, 0x9B, 0x56, 0x1C, 0xB5,
0xA4, 0xA5, 0x0D, 0x61, 0x52, 0x32,
    0x0C, 0x4E, 0xE8, 0xCA, 0x4B, 0x9E, 0xA2, 0x4D, 0x35, 0x0E, 0xB1, 0xA1, 0x5B, //
NOLINT(readability-magic-numbers)
    0x5B, 0xD8, 0xC1, 0x93, 0x28, 0x60, 0x90, 0x18, 0x37, 0x88, 0x66, 0xD4, 0xED,
0x37, 0x6B, 0xEC, 0x48, 0xFF, 0x3D,
    0xFB, 0x99, 0xB7, 0xEF, 0x3C, 0x8D, 0x25, 0xE9, 0xF3, 0xCF, 0x02, 0x7C, 0x7D, //
NOLINT(readability-magic-numbers)
    0xB9, 0x4D, 0x2F, 0x2F, 0x1E, 0x30, 0x48, 0x54, 0x18, 0xE4, 0x51, 0xD5, 0xC1,
0xB7, 0x3B, 0x0D, 0xE4, 0xB4, 0x19,
    0x7B, 0x9B, 0xF4, 0x35, 0x82, 0xCC, 0x91, 0x4C, 0x10, 0xE9, 0xB6, 0xF1, 0x0A, //
NOLINT(readability-magic-numbers)
    0x23, 0xEB, 0x7D, 0x51, 0x47, 0x36, 0xFE, 0x13, 0xAF, 0x3C, 0x23, 0x9F, 0x7E,
0xFC, 0xCF, 0x7A, 0x7C, 0x2D, 0xDB,
    0xD9, 0xDA, 0xEB, 0xF7, 0xB5, 0x6B, 0x8D, 0xE0, 0x18, 0x9A, 0x5B, 0xB7, 0x0A, //
NOLINT(readability-magic-numbers)
    0xA3, 0x4E, 0xE1, 0xB7, 0xF7, 0xD1, 0x94, 0xD5, 0x7A, 0xD3, 0x27, 0xE2, 0x1F,
0x3A, 0xEB, 0xF0, 0x83, 0x10, 0x52,
    0x51, 0x5F, 0x58, 0xF8, 0x81, 0x42, 0x48, 0x83, 0x2D, 0xF0, 0xA9, 0x7D, 0x79, //
NOLINT(readability-magic-numbers)
    0x2B, 0xF1, 0x68, 0xC2, 0x22, 0xC0, 0x0C, 0x72, 0x63, 0x37, 0xBF, 0xEC, 0x72,
0x97, 0xD4, 0xA5, 0x91, 0x2E, 0x1F,
    0xA3, 0x78, 0x9A, 0xCE, 0xFE, 0x27, 0x7F, 0x2B, 0x85, 0x7D, 0x22, 0x2C, 0x0D, //
NOLINT(readability-magic-numbers)
    0x1E, 0x10, 0xB7, 0xFF, 0x9A, 0xA7, 0x99, 0xD2, 0xB9, 0x40, 0x53, 0xB3, 0xA9,
0x52, 0x5D, 0xBD, 0xC8, 0x12, 0x8D,
    0x39, 0xD7, 0x97, 0x03, 0xD2, 0x80, 0x21, 0xC3, 0xA7, 0x8B, 0xE3, 0x3D, 0xF0, //
NOLINT(readability-magic-numbers)
    0x4D, 0x4C, 0x4D, 0xC4, 0xC7, 0xE5, 0xE4, 0x35, 0x75, 0xAA, 0x45, 0x3B, 0x9C,
0x64, 0xC1, 0x94, 0x6E, 0x15, 0x0A,
    0xE8, 0x84, 0xCD, 0xFC, 0x7A, 0xBC, 0x5C, 0x8C, 0xA8, 0x95, 0x07, 0x79, 0x4E, //
NOLINT(readability-magic-numbers)
};
```

```
void psa_rsa2048_pem_format_import_example (void)
{
/* This example imports an RSA2048 keypair of PEM format, performs signing and
verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length = 0U;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* Import the private key (This includes both the private and public key) */
    mbedtls_pk_context ctx_rsa;
    mbedtls_pk_init(&ctx_rsa);
 /* For keys other than ASN format, mbedtls_pk_parse_key and
mbedtls_pk_import_into_psa APIs must be called instead of psa_import_key */
 if (PSA_SUCCESS !=
       mbedtls_pk_parse_key(&ctx_rsa, RSAKeydata, sizeof(RSAKeydata), NULL, 0,
mbedtls_psa_get_random,
                            MBEDTLS_PSA_RANDOM_STATE))
```

```
   {
      debugger_break();
   }
 if (PSA_SUCCESS != mbedtls_pk_import_into_psa(&ctx_rsa, &attributes, &key_handle))
   {
      debugger_break();
   }
   mbedtls_pk_free(&ctx_rsa);
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
   {
      debugger_break();
   }
 /* Sign message using the key */
 if (PSA_SUCCESS !=
      psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature, sizeof(signature),
                    &signature_length))
   {
      debugger_break();
   }
 if (0 != memcmp(signature, &RSAExpectedSignature, sizeof(RSAExpectedSignature)))
   {
      debugger_break();
   }
 /* Verify the signature */
 if (PSA_SUCCESS !=
      psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), RSAExpectedSignature,
 sizeof(RSAExpectedSignature)))
   {
      debugger_break();
   }
 /* Destroy the key and handle */
```

```
if (PSA_SUCCESS != psa_destroy_key(key_handle))
   {
      debugger_break();
   }
   mbedtls_psa_crypto_free();
mbedtls_platform_teardown(&ctx);
}
```

### Function Documentation

#### ◆ mbedtls_platform_setup()

int mbedtls_platform_setup ( mbedtls_platform_context * *ctx*)

This function initializes the SCE and the TRNG. It **must** be invoked before the crypto library can be used. This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined.

Example:

```
mbedtls_platform_context ctx = {0};
/* Setup the platform; initialize the SCE and the TRNG */
if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
```

**Return values**

| | |
|---|---|
| 0 | Initialization was successful. |
| MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED | SCE Initialization error. |

### ◆ mbedtls_platform_teardown()

void mbedtls_platform_teardown ( mbedtls_platform_context * *ctx*)

This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined. It is intended to de-initialize any items that were initialized in the mbedtls_platform_setup() function, but currently is only a placeholder function.

Example:

```
/* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
mbedtls_platform_teardown(&ctx);
```

**Return values**

| | |
|---|---|
| N/A | |

### ◆ RM_PSA_CRYPTO_TRNG_Read()

fsp_err_t RM_PSA_CRYPTO_TRNG_Read ( uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes*, uint32_t * *p_num_gen_bytes* )

Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Random number generation successful |
| FSP_ERR_ASSERTION | NULL input parameter(s). |
| FSP_ERR_CRYPTO_UNKNOWN | An unknown error occurred. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- s_generate_16byte_random_data

## 5.2.15.4 Ocrypto (rm_ocrypto_port)

Modules » Security

**Functions**

| | |
|---|---|
| fsp_err_t | RM_OCRYPTO_PORT_Init () |
| | Initialize the SCE. More... |

| | |
|---|---|
| fsp_err_t | RM_OCRYPTO_PORT_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes)<br><br>Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer. More... |

### Detailed Description

Ocrypto software only crypto on RA2E1.

# Overview

Ocrypto is designed as a small footprint software crypto implementation to be used on resource constrained devices. The software only module is available in FSP only on RA2E1.

*Note*

***The Ocrypto port module does not provide any crypto interfaces to the user. See Ocrypto Licensed Functions.***

### Hardware Overview

Limited to RA2E1 only.

### Features

For features supported by the software-only version, refer to the Ocrypto documentation.

# Configuration

### Build Time Configurations for rm_ocrypto_trng_port

The following build time configurations are defined in fsp_cfg/rm_ocrypto_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### AES Configuration

To enable hardware acceleration for the AES-EBC, CBC, and CTR operations, choose Ocrypto (H/W Accelerated) from the stack options. Be sure to initialize the hardware

# Usage Notes

### Hardware Initialization

Invoke RM_OCRYPTO_PORT_Init() to initialize the hardware before using Ocrypto if the TRNG or if H/W Acceleration is to be used.

### Random Number Generation

Use the RM_OCRYPTO_PORT_TRNG_Read() function to obtain random data from the TRNG hardware. If purely random data is sufficient for the application, then RM_OCRYPTO_PORT_TRNG_Read() can be used directly instead.

### Memory Usage

Ocrypto does not use dynamic allocation so there is no heap requirement.

A minimum stack of 0x800 is required to use Ocrypto.

### Side Channel Safety Statement

See Ocrypto Side Channel Safety Statement.

### Function Overview

### Limitations

### Usage with RA2E1

Ocrypto can only be used on RA2E1.

### Ocrypto

Ocrypto does provide any API to generate keys. It is the responsibility of the caller to provide generated keys.

### AES Reentrancy

Use the S/W only stack if AES reentrancy is required in your application.

# Examples

### Ocrypto Library Examples

For examples of how to use Ocrypto APIs, see Ocrypto test code under ra/oberon/ocrypto/tests/. These tests are not meant to be built or run in FSP and have been compiled out by the macro OCRYPTO_REFERENCE_TEST_CODE_DOES_NOT_BUILD by default.

### TRNG Example

This is an example on using Ocrypto to generate a random number

```
#define TEST_SECRET_KEY_SIZE 32U
#define TEST_PUBLIC_KEY_SIZE 64U
void ocrypto_trng_example (void)
{
 fsp_err_t status       = FSP_SUCCESS;
    uint8_t   ecdsa_msg[3] = "abc";
```

```
    uint8_t   secret_key[TEST_SECRET_KEY_SIZE];
    uint8_t   session_key[TEST_SECRET_KEY_SIZE];
    uint8_t   ecdsa_public_key[TEST_PUBLIC_KEY_SIZE];
    uint8_t   signature[TEST_PUBLIC_KEY_SIZE];
 /* Setup the platform; initialize the crypto engine. */
    status = RM_OCRYPTO_PORT_Init();
 if (FSP_SUCCESS != status)
    {
       assert(0);
    }
 /* Read random data from the TRNG to use as secret key. */
    status = RM_OCRYPTO_PORT_TRNG_Read(secret_key, sizeof(secret_key));
 if (FSP_SUCCESS != status)
    {
       assert(0);
    }
 if (0 != ocrypto_ecdsa_p256_public_key(ecdsa_public_key, secret_key))
    {
       assert(0);
    }
 /* Read random data from the TRNG to use as session key */
    status = RM_OCRYPTO_PORT_TRNG_Read(session_key, sizeof(session_key));
 if (FSP_SUCCESS != status)
    {
       assert(0);
    }
 if (0 != ocrypto_ecdsa_p256_sign(signature, ecdsa_msg, sizeof(ecdsa_msg),
secret_key, session_key))
    {
       assert(0);
    }
 if (0 != ocrypto_ecdsa_p256_verify(signature, ecdsa_msg, sizeof(ecdsa_msg),
ecdsa_public_key))
    {
```

```
      assert(0);
   }
/* Operation successful. */
while (1)
   {
      ;
   }
}
```

### Function Documentation

#### ◆ RM_OCRYPTO_PORT_Init()

| fsp_err_t RM_OCRYPTO_PORT_Init ( ) |
|---|
| Initialize the SCE.<br><br>**Return values**<br>FSP_SUCCESS — SCE initialized successfully<br><br>**Returns**<br>See Common Error Codes or functions called by this function for other possible return codes. |

#### ◆ RM_OCRYPTO_PORT_TRNG_Read()

| fsp_err_t RM_OCRYPTO_PORT_TRNG_Read ( uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes* ) |
|---|
| Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer.<br><br>**Parameters**<br>[out] p_rngbuf — Pointer to buffer to store generated random number<br>[in] num_req_bytes — How many bytes of random numbers to generate<br><br>**Return values**<br>FSP_SUCCESS — Random number generation successful<br>FSP_ERR_ASSERTION — NULL input parameter(s).<br>FSP_ERR_CRYPTO_UNKNOWN — An unknown error occurred. |

### 5.2.15.5 Renesas Secure IP (r_rsip_e11a_protected_cavp)

Modules » Security

Driver for the CAVP Certified Renesas Secure IP (RSIP-E11A) on RA MPUs.

# Version

v2.0.0

The version that appears on the Components tab of the FSP Configuration editor is v2.0.0+fsp.<fsp_version>. The <fsp_version> metadata reflects tooling support files only and does not indicate any changes to the CAVP certified code. See Module Versioning for more information about component versioning in FSP.

# Functions

The specifications of this module comply with the Renesas Secure IP (r_rsip_protected) of FSP v6.0.0. For more information about the features of this module, see the "Renesas Flexible Software Package (FSP) v6.0.0 User's Manual." The documentation can be obtained from the following link:

See also
https://www.renesas.com/en/software-tool/flexible-software-package-fsp

# Overview

This module provides RSIP-E11A CAVP Certified functions.

### HW Overview

RSIP-E11A Protected Mode CAVP certification has been obtained using the RA4L1 as the target processor.

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E11A | RA4L1 |

### Features

The RSIP module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/256bit
      - CBC 128/256bit
      - CTR 128/256bit
      - GCM 128/256bit

        - CCM 128/256bit
    - Asymmetric Encryption/Decryption
        - ECC
            - ECDSA
                - secp256r1
    - Key Agreement
        - ECDH
            - secp256r1
    - Key Derivation
        - Key Derivation Function (KDF) using SHA
            - SHA-256bit
        - Key Import by derived keying material
        - IV output by derived keying material
    - Public Key Infrastructure (PKI)
        - Certificate Verification
            - ECDSA
                - secp256r1
        - Key Import from Certificate
            - ECC
                - secp256r1
    - Hash Functions
        - SHA-2
            - SHA-224/256
    - RFC3394 Key Wrap/Unwrap
- Message Authentication Code
    - HMAC-SHA224/256bit
    - AES-CMAC 128/256bit
- Key Manage
    - Key Support
        - AES 128/256bit
        - ECC
            - secp256r1
        - HMAC-SHA224/256bit
- Random Number Generate

### Supported algorithms

The following algorithms are available on each devices.

- RSIP-E11A: RA4L1

### Symmetric Key

| Key type (rsip_key_type_t) | RSIP-E11A |
|---|---|
| RSIP_KEY_TYPE_AES_128 | Supported |
| RSIP_KEY_TYPE_AES_192 | Not Supported |
| RSIP_KEY_TYPE_AES_256 | Supported |
| RSIP_KEY_TYPE_XTS_AES_128 | Not Supported |
| RSIP_KEY_TYPE_XTS_AES_256 | Not Supported |
| RSIP_KEY_TYPE_HMAC_SHA224 | Supported |
| RSIP_KEY_TYPE_HMAC_SHA256 | Supported |

| | |
|---|---|
| RSIP_KEY_TYPE_HMAC_SHA384 | Not Supported |
| RSIP_KEY_TYPE_HMAC_SHA512 | Not Supported |

### Asymmetric Key

| Key pair type (rsip_key_pair_type_t) | Public key type (rsip_key_type_t) | Private key type (rsip_key_type_t) | RSIP-E11A |
|---|---|---|---|
| RSIP_KEY_PAIR_TYPE_ECC_SECP256R1 | RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP384R1 | RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP521R1 | RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP256K1 | RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP256R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP384R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP512R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_RSA_2048 | RSIP_KEY_TYPE_RSA_2048_PUBLIC | RSIP_KEY_TYPE_RSA_2048_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_RSA_3072 | RSIP_KEY_TYPE_RSA_3072_PUBLIC | RSIP_KEY_TYPE_RSA_3072_PRIVATE | Not Supported |
| RSIP_KEY_PAIR_TYPE_RSA_4096 | RSIP_KEY_TYPE_RSA_4096_PUBLIC | RSIP_KEY_TYPE_RSA_4096_PRIVATE | Not Supported |

### AES Block Cipher Mode of Operation

| Cipher mode (rsip_aes_cipher_mode_t) | RSIP-E11A |
|---|---|
| RSIP_AES_CIPHER_MODE_ECB_ENC | Supported |
| RSIP_AES_CIPHER_MODE_ECB_DEC | Supported |
| RSIP_AES_CIPHER_MODE_CBC_ENC | Supported |
| RSIP_AES_CIPHER_MODE_CBC_DEC | Supported |
| RSIP_AES_CIPHER_MODE_CTR | Supported |
| RSIP_AES_CIPHER_MODE_XTS_ENC | Not Supported |
| RSIP_AES_CIPHER_MODE_XTS_DEC | Not Supported |

| AEAD mode (rsip_aes_aead_mode_t) | RSIP-E11A |
|---|---|

| | |
|---|---|
| RSIP_AES_AEAD_MODE_GCM_ENC | Supported |
| RSIP_AES_AEAD_MODE_GCM_DEC | Supported |
| RSIP_AES_AEAD_MODE_CCM_ENC | Supported |
| RSIP_AES_AEAD_MODE_CCM_DEC | Supported |

| MAC mode (rsip_aes_mac_mode_t) | RSIP-E11A |
|---|---|
| RSIP_AES_MAC_MODE_CMAC | Supported |

**Hash Function**

| Hash Function | RSIP-E11A |
|---|---|
| RSIP_HASH_TYPE_SHA224 | Supported |
| RSIP_HASH_TYPE_SHA256 | Supported |
| RSIP_HASH_TYPE_SHA384 | Not Supported |
| RSIP_HASH_TYPE_SHA512 | Not Supported |
| RSIP_HASH_TYPE_SHA512_224 | Not Supported |
| RSIP_HASH_TYPE_SHA512_256 | Not Supported |

**Key Derivation Function**

| Key Derivation Function | RSIP-E11A |
|---|---|
| SHA-256 | Supported |
| SHA-384 | Not Supported |
| HMAC-SHA256 | Not Supported |
| HMAC-SHA384 | Not Supported |
| HMAC-SHA512 | Not Supported |

# Configuration

### Build Time Configurations for r_rsip_e11a_protected_cavp

The following build time configurations are defined in fsp_cfg/r_rsip_cavp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the |

| | | | |
|---|---|---|---|
| | | | functions for which it works effectively. |
| AES-192 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-128 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused |

| | | | |
|---|---|---|---|
| | | | algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | Enabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the |

| | | | |
|---|---|---|---|
| | | | function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP384R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP256R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |

| | | | |
|---|---|---|---|
| ECC BRAINPOOLP384R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC EDWARDS25519 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-3072 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| RSA-4096 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Security > RSIP-E11A Protected Mode (CAVP Certified)**

This module can be added to the Stacks tab via New Stack > Security > RSIP-E11A Protected Mode (CAVP Certified).

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_rsip | Module name. |

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Getting Started: Creating a RSIP Protected Mode Project

Start by creating a new project in e² studio or RASC. On the Stacks tab, add New > Security > RSIP-E11A Protected Mode (CAVP Certified).

### State Transition

This driver has the following states.

| State Name | Details |
|---|---|
| STATE_INITIAL | Driver is not open. |
| STATE_MAIN | Ready to Compute. |
| STATE_AES | Computing AES (unauthenticated cipher) |
| STATE_AES_AEAD | Computing AES (AEAD). |
| STATE_AES_MAC | Computing AES (MAC). |
| STATE_SHA | Computing SHA. |
| STATE_HMAC | Computing HMAC. |
| STATE_KDF_SHA | Computing KDF SHA. |
| STATE_KDF_HMAC | Computing KDF HMAC. |

There are two types of APIs provided by the RSIP driver for accelerating cryptographic operations: those that provide cryptographic operations in a single API and those that provide them in multiple APIs. In this document, the former is referred to as single-part operations and the latter as multi-part operations. Each corresponds to the following algorithms:

- Single-part operations: DRBG, Key management, ECC, SHA, HMAC
- Multi-part operations: AES, SHA, HMAC

Multi-part operations are APIs which split a single cryptographic operation into a sequence of separate steps (e.g. Init-Update-Finish). This enables fine control over the configuration of the cryptographic operation, and allows the message data to be processed in fragments instead of all at once.

Due to the above characteristics, it is necessary to manage the operational states in the multi-part

operations.

# Examples

### AES cipher Example

This is an example of AES-CBC encryption/decryption.

```
void r_rsip_example_aes ()
{
 fsp_err_t            err;
 static const uint8_t plain[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
    };
    uint8_t iv[RSIP_PRV_BYTE_SIZE_AES_BLOCK] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t plain_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
```

```
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, iv);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, plain, cipher_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_DEC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, cipher_calculated, plain_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl,
                                   cipher_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   plain_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-GCM Example

This is an example of AES-GCM encryption/decryption.

```
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_NONCE_LEN (12)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (8)
void r_rsip_example_gcm ()
{
 fsp_err_t            err;
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)               = {0};
    uint8_t tag[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                  = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
```

```
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) */
    err            = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
```

```
    input_length = 0;
    output_length = 0;
 /* Input ciphertext (multi-shot) */
    input_length_tmp = 10;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
    input_length_tmp = GCM_TEXT_LEN - input_length;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /*
  * Output remaining text and verify the tag
  * Attention: Do not use the output plaintext by R_RSIP_AES_Cipher_DecryptUpdate()
  * before R_RSIP_AES_Cipher_DecryptFinal() returns FSP_SUCCESS.
  * The integrity is checked in this function.
  */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
```

```
}
```

### AES-GMAC Example

This is an example of AES-GMAC signature/verification.

```
#define GMAC_NONCE_LEN (12)
#define GMAC_AAD_LEN (8)
#define GMAC_TAG_LEN (16)
void r_rsip_example_gmac ()
{
 fsp_err_t err;
 static const uint8_t aad[GMAC_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GMAC_NONCE_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t tag[GMAC_TAG_LEN] BSP_ALIGN_VARIABLE(4)    = {0};
    uint8_t output_dummy_buf[1] = {0};
    uint32_t output_length       = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
```

```
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Initialize GMAC signature */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC generation */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, output_dummy_buf, &output_length,
tag);
    assert(FSP_SUCCESS == err);
 /*Initialize GMAC verification*/
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC verification*/
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, output_dummy_buf, &output_length, tag,
GMAC_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CCM Example

This is an example of AES-CCM encryption/decryption.

```
#define CCM_NONCE_LEN (12)
#define CCM_ADATA_LEN (20)
```

```
#define CCM_INPUT_DATA_LEN (24)
#define CCM_MAC_LEN (8)
void r_rsip_example_ccm ()
{
 fsp_err_t err;
 static const uint8_t nonce[CCM_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B};
 static const uint8_t adata[CCM_ADATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
       0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13
    };
 static const uint8_t plain[CCM_INPUT_DATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29,
       0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33,
       0x34, 0x35, 0x36, 0x37
    };
    uint8_t   cipher_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   plain_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   mac[CCM_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length_tmp = 0;
    uint8_t * p_output          = NULL;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
```

```
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_ENC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(cipher_calculated[output_length_tmp]);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, p_output, &output_length_tmp, mac);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_DEC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
```

```
 // err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
plain_calculated, &output_length_tmp);
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                 cipher_calculated,
                                 CCM_INPUT_DATA_LEN,
                                 plain_calculated,
                                 &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(plain_calculated[output_length_tmp]);
 /* Output remaining text and verify the tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, p_output, &output_length_tmp, mac,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CMAC Example

This is an example of AES-CMAC signature/verification.

```
#define CMAC_MESSAGE_LEN (10)
#define CMAC_MAC_LEN (16)
void r_rsip_example_cmac ()
{
 fsp_err_t err;
 static const uint8_t message[CMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0xf0, 0x8f, 0x89, 0x08, 0x75, 0xe1, 0x39, 0x48, 0x04, 0x89
    };
    uint8_t mac[CMAC_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t           wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
```

```
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for encrypt */
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_MAC_SignFinish(&g_rsip_ctrl, mac);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for decrypt */
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Verify the tag */
    err = R_RSIP_AES_MAC_VerifyFinish(&g_rsip_ctrl, mac, CMAC_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ECC Example

This is an example of ECDSA signature generation/verification.

```
#define SECP256R1_KEY_SIZE (32)
void r_rsip_example_ecc ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
    uint8_t hash[SECP256R1_KEY_SIZE] BSP_ALIGN_VARIABLE(4)          = {0};
    uint8_t signature[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
```

```
    assert(FSP_SUCCESS == err);
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_ECDSA_Verify(&g_rsip_ctrl, &wrapped_public_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HASH Example

This is an example of calculating the SHA-256 message digest.

```
#define SHA256_HASH_SIZE (32)
void r_rsip_example_hash ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE (4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
       0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    };
 static const uint8_t hash[] BSP_ALIGN_VARIABLE (4) =
    {
       0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
```

```
      0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
    };
    uint8_t digest_single[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t digest_multi[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate SHA-256 digests with single-part API */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), digest_single);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_single, hash, SHA256_HASH_SIZE));
 /* (2) Calculate SHA-256 digests with multi-part API */
    err = R_RSIP_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Update(&g_rsip_ctrl, message, sizeof(message));
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Finish(&g_rsip_ctrl, digest_multi);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_multi, hash, SHA256_HASH_SIZE));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HMAC Example

This is an example of HMAC signature/verification.

```
#define HMAC_MESSAGE_LEN (34)
#define HMAC_HASH_LEN (32)
void r_rsip_example_hmac ()
```

```
{
 fsp_err_t err;
 static const uint8_t message[HMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x53, 0x61, 0x6D, 0x70, 0x6C, 0x65, 0x20, 0x6D,
       0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x20, 0x66,
       0x6F, 0x72, 0x20, 0x6B, 0x65, 0x79, 0x6C, 0x65,
       0x6E, 0x3C, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x6C,
       0x65, 0x6E
    };
    uint8_t hash_1[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t hash_2[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a HMAC-SHA 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to finish */
    err = R_RSIP_HMAC_Compute(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1);
    assert(FSP_SUCCESS == err);
 /* (2) Verify HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to verify */
    err = R_RSIP_HMAC_Verify(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
```

```
hash_1, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* (3) Calculate HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC generation */
    err = R_RSIP_HMAC_SignFinish(&g_rsip_ctrl, hash_2);
    assert(FSP_SUCCESS == err);
 /* (4) Verify HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC verification */
    err = R_RSIP_HMAC_VerifyFinish(&g_rsip_ctrl, hash_2, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare hash_1 and hash_2 */
    assert(0 == memcmp(hash_1, hash_2, HMAC_HASH_LEN));
}
```

### PKI Example

This is an example that verifies the public key certificate, imports the peer's public key, and calculates the shared secret.

```
#define SHA256_HASH_SIZE (32)
```

```
#define SECP256R1_KEY_SIZE (32)
/* Simplified public key certificate structure */
typedef struct st_cert
{
    uint8_t public_key_param_qx[SECP256R1_KEY_SIZE];
    uint8_t public_key_param_qy[SECP256R1_KEY_SIZE];
} cert_t;
void r_rsip_example_pki ()
{
 fsp_err_t err;
 /* Prepare data area for generating certificate and shared secret */
    cert_t               cert;
    uint8_t              raw_public_key[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t              hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t              signature[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4)      = {0};
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare wrapped key data area for CA (Virtually prepared for local validation) */
    uint8_t
wrapped_ca_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_ca_public_key_value
    };
    uint8_t
wrapped_ca_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_private_key =
```

```
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_ca_private_key_value
    };
 /* Prepare wrapped key data area for peer (Virtually prepared for local validation)
*/
    uint8_t
wrapped_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_peer_public_key_value
    };
    uint8_t
wrapped_peer_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_peer_private_key_value
    };
 /* Prepare wrapped key data area for Local device */
    uint8_t
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
```

```
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 /* Prepare wrapped key data area for verified peer public key */
    uint8_t wrapped_verified_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
 RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_verified_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_verified_peer_public_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate key pairs for CA, peer, and Local device */
 /* Generate a key pair of secp256r1 for CA (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_ca_public_key,
&wrapped_ca_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for peer (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_peer_public_key,
&wrapped_peer_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device (Assume product with RSIP
driver) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
```

```
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
/* CA process :
 * In the following, generate certificate and signature of peer public key */
 /* Export raw peer public key (Qx|Qy) and store it in cert */
    err = R_RSIP_PublicKeyExport(&wrapped_peer_public_key, raw_public_key);
    assert(FSP_SUCCESS == err);
    memcpy(cert.public_key_param_qx, &raw_public_key[0], SECP256R1_KEY_SIZE);
    memcpy(cert.public_key_param_qy, &raw_public_key[SECP256R1_KEY_SIZE],
SECP256R1_KEY_SIZE);
 /* Generate the hash of cert and store it in hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, (uint8_t *) &cert,
sizeof(cert), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature of hash with CA private key and store it in signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_ca_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
/* Local device process :
 * In the following, verify the CA certificate, import peer public key, and generate
shared secret */
 /* Verify the signature of cert with CA public key */
    err = R_RSIP_PKI_ECDSA_CertVerify(&g_rsip_ctrl, &wrapped_ca_public_key, hash,
signature);
    assert(FSP_SUCCESS == err);
 /* Import verified peer public key with wrapped format */
    err = R_RSIP_PKI_CertKeyImport(&g_rsip_ctrl,
                                   (uint8_t *) &cert,
 sizeof(cert),
                                   cert.public_key_param_qx,
                                   SECP256R1_KEY_SIZE,
                                   cert.public_key_param_qy,
                                   SECP256R1_KEY_SIZE,
 RSIP_HASH_TYPE_SHA256,
                                   &wrapped_verified_peer_public_key);
```

```
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the Local private key and the verified peer public
key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl,
                               &wrapped_local_private_key,
                               &wrapped_verified_peer_public_key,
                               &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### KDF-SHA Example

This is an example of One-Step Key Derivation (NIST SP 800-56C) using KDF-SHA.

```
#define COUNTER_BYTE_LEN (4)
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (64)
#define CONCAT_DKM_BYTE_SIZE KDF_SHA256_DKM_BYTE_SIZE * 2
#define WRAPPED_IV_BYTE_SIZE (36)
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (16)
void r_rsip_example_kdf_sha ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
```

```
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_DKM_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-SHA operation */
    uint32_t             counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
```

```
0x0d, 0x0e, 0x0f,
    };
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* One-Step Key Derivation (NIST SP 800-56C)
```

```
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter = 0x00000000;
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM1 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
```

```
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM2 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_DKM_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In One-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   16,
                                   NULL,
                                   wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
```

```
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, cipher_calculated, GCM_TEXT_LEN,
plain_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
```

```
/* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### 5.2.15.6 Renesas Secure IP (r_rsip_e51a_protected_cavp)

Modules » Security

Driver for the CAVP Certified Renesas Secure IP (RSIP-E51A) on RA MPUs.

# Version

v2.1.0

The version that appears on the Components tab of the FSP Configuration editor is v2.1.0+fsp.<fsp_version>. The <fsp_version> metadata reflects tooling support files only and does not indicate any changes to the CAVP certified code. See Module Versioning for more information about component versioning in FSP.

# Functions

The specifications of this module comply with the Renesas Secure IP (r_rsip_protected) of FSP v6.1.0. For more information about the features of this module, see the "Renesas Flexible Software Package (FSP) v6.1.0 User's Manual." The documentation can be obtained from the following link:

See also
https://www.renesas.com/en/software-tool/flexible-software-package-fsp

# Overview

This module provides RSIP-E51A CAVP Certified functions.

### HW Overview

RSIP-E51A Protected Mode CAVP certification has been obtained using the RA8M1 as the target processor.

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E51A | RA8M1, RA8D1, RA8T1 |

### Features

The RSIP module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/192/256bit
      - CBC 128/192/256bit
      - CTR 128/192/256bit
      - GCM 128/192/256bit
      - CCM 128/192/256bit
      - XTS 128/256bit
  - Asymmetric Encryption/Decryption
    - RSA
      - RSAES-PKCS1-V1_5 2048/3072/4096bit
      - RSAES-OAEP 2048/3072/4096bit
      - RSASSA-PKCS1-V1_5 2048/3072/4096bit
      - RSASSA-PSS 2048/3072/4096bit
    - ECC
      - ECDSA
        - secp256r1/secp384r1/secp521r1
        - secp256k1
        - brainpool P256r1/P384r1/P512r1
      - EdDSA
        - Ed25519
  - Key Agreement
    - ECDH
      - secp256r1/secp384r1/secp521r1
        - brainpool P256r1/P384r1
  - Key Derivation
    - Key Derivation Function (KDF) using SHA
      - SHA-256/384bit
    - Key Derivation Function (KDF) using HMAC
      - HMAC-SHA256/384/512bit
    - Key Import by derived keying material
    - IV output by derived keying material
  - Public Key Infrastructure (PKI)
    - Certificate Verification
      - ECDSA
        - secp256r1/secp384r1/secp521r1
        - brainpool P256r1/P384r1
    - Key Import from Certificate
      - ECC
        - secp256r1/secp384r1/secp521r1
        - brainpool P256r1/P384r1
  - Hash Functions
    - SHA-2
      - SHA-224, 256, 384, 512, 512/224, 512/256
  - RFC3394 Key Wrap/Unwrap
- Message Authentication Code
  - HMAC-SHA224/256/384/512bit
  - AES-CMAC 128/192/256bit
- Key Manage
  - Key Support
    - AES 128/192/256bit
    - RSA 2048/3072/4096bit
    - ECC
      - secp256r1/secp384r1/secp521r1

        - secp256k1
        - brainpool P256r1/P384r1/P512r1
    - HMAC-SHA224/256/384/512bit
- Random Number Generate
- Secure boot for FSBL
    - Manifest verification and MAC generation (Refer to Application Note R11AN0774.)

### Supported algorithms

The following algorithms are available on each devices.

- RSIP-E51A: RA8M1, RA8D1, RA8T1

### Symmetric Key

| Key type (rsip_key_type_t) | RSIP-E51A |
|---|---|
| RSIP_KEY_TYPE_AES_128 | Supported |
| RSIP_KEY_TYPE_AES_256 | Supported |
| RSIP_KEY_TYPE_XTS_AES_128 | Supported |
| RSIP_KEY_TYPE_XTS_AES_256 | Supported |
| RSIP_KEY_TYPE_HMAC_SHA224 | Supported |
| RSIP_KEY_TYPE_HMAC_SHA256 | Supported |
| RSIP_KEY_TYPE_HMAC_SHA384 | Supported |
| RSIP_KEY_TYPE_HMAC_SHA512 | Supported |

### Asymmetric Key

| Key pair type (rsip_key_pair_type_t) | Public key type (rsip_key_type_t) | Private key type (rsip_key_type_t) | RSIP-E51A |
|---|---|---|---|
| RSIP_KEY_PAIR_TYPE_ECC_SECP256R1 | RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP384R1 | RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP521R1 | RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_SECP256K1 | RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP256R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP384R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_ECC_BRAINPOOLP512R1 | RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE | Supported |

| | | | |
|---|---|---|---|
| RSIP_KEY_PAIR_TYPE_RSA_2048 | RSIP_KEY_TYPE_RSA_2048_PUBLIC | RSIP_KEY_TYPE_RSA_2048_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_RSA_3072 | RSIP_KEY_TYPE_RSA_3072_PUBLIC | RSIP_KEY_TYPE_RSA_3072_PRIVATE | Supported |
| RSIP_KEY_PAIR_TYPE_RSA_4096 | RSIP_KEY_TYPE_RSA_4096_PUBLIC | RSIP_KEY_TYPE_RSA_4096_PRIVATE | Supported |

### AES Block Cipher Mode of Operation

| Cipher mode (rsip_aes_cipher_mode_t) | RSIP-E51A |
|---|---|
| RSIP_AES_CIPHER_MODE_ECB_ENC | Supported |
| RSIP_AES_CIPHER_MODE_ECB_DEC | Supported |
| RSIP_AES_CIPHER_MODE_CBC_ENC | Supported |
| RSIP_AES_CIPHER_MODE_CBC_DEC | Supported |
| RSIP_AES_CIPHER_MODE_CTR | Supported |
| RSIP_AES_CIPHER_MODE_XTS_ENC | Supported |
| RSIP_AES_CIPHER_MODE_XTS_DEC | Supported |

| AEAD mode (rsip_aes_aead_mode_t) | RSIP-E51A |
|---|---|
| RSIP_AES_AEAD_MODE_GCM_ENC | Supported |
| RSIP_AES_AEAD_MODE_GCM_DEC | Supported |
| RSIP_AES_AEAD_MODE_CCM_ENC | Supported |
| RSIP_AES_AEAD_MODE_CCM_DEC | Supported |

| MAC mode (rsip_aes_mac_mode_t) | RSIP-E51A |
|---|---|
| RSIP_AES_MAC_MODE_CMAC | Supported |

### Hash Function

| Hash Function (rsip_hash_type_t) | RSIP-E51A |
|---|---|
| RSIP_HASH_TYPE_SHA224 | Supported |
| RSIP_HASH_TYPE_SHA256 | Supported |
| RSIP_HASH_TYPE_SHA384 | Supported |
| RSIP_HASH_TYPE_SHA512 | Supported |

### Key Derivation Function

| Key Derivation Function | RSIP-E51A |
|---|---|
| SHA-256 | Supported |
| SHA-384 | Supported |
| HMAC-SHA256 | Supported |
| HMAC-SHA384 | Supported |

| | |
|---|---|
| HMAC-SHA512 | Supported |

# Configuration

### Build Time Configurations for r_rsip_e51a_protected_cavp

The following build time configurations are defined in fsp_cfg/r_rsip_cavp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-192 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| XTS-AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it |

| | | | |
|---|---|---|---|
| | | | works effectively. |
| ECC EDWARDS25519 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-3072 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-4096 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This |

| | | | |
|---|---|---|---|
| | | | configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size |

| | | | |
|---|---|---|---|
| | | | can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | • Enabled | Enabled | If the application uses |

| | | | |
|---|---|---|---|
| | • Disabled | | only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Security > RSIP-E51A Protected Mode (CAVP Certified)**

This module can be added to the Stacks tab via New Stack > Security > RSIP-E51A Protected Mode (CAVP Certified).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_rsip | Module name. |

**Clock Configuration**

This module does not require a specific clock configuration.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

**Getting Started: Creating a RSIP Protected Mode Project**

Start by creating a new project in e² studio or RASC. On the Stacks tab, add New > Security > RSIP-E51A Protected Mode (CAVP Certified).

**State Transition**

This driver has the following states.

| State Name | Details |
|---|---|
| STATE_INITIAL | Driver is not open. |
| STATE_MAIN | Ready to Compute. |
| STATE_AES | Computing AES (unauthenticated cipher) |
| STATE_AES_AEAD | Computing AES (AEAD). |
| STATE_AES_MAC | Computing AES (MAC). |
| STATE_SHA | Computing SHA. |
| STATE_HMAC | Computing HMAC. |
| STATE_KDF_SHA | Computing KDF SHA. |
| STATE_KDF_HMAC | Computing KDF HMAC. |

There are two types of APIs provided by the RSIP driver for accelerating cryptographic operations: those that provide cryptographic operations in a single API and those that provide them in multiple APIs. In this document, the former is referred to as single-part operations and the latter as multi-part operations. Each corresponds to the following algorithms:

- Single-part operations: DRBG, Key management, ECC, RSA, SHA, HMAC
- Multi-part operations: AES, SHA, HMAC

Multi-part operations are APIs which split a single cryptographic operation into a sequence of separate steps (e.g. Init-Update-Finish). This enables fine control over the configuration of the cryptographic operation, and allows the message data to be processed in fragments instead of all at once.

Due to the above characteristics, it is necessary to manage the operational states in the multi-part operations.

# Examples

### AES cipher Example

This is an example of AES-CBC encryption/decryption.

```
void r_rsip_example_aes ()
{
 fsp_err_t            err;
 static const uint8_t plain[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
```

```
0x6f, 0x6e, 0x00
    };
    uint8_t iv[RSIP_PRV_BYTE_SIZE_AES_BLOCK] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t plain_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, iv);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, plain, cipher_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
```

```
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_DEC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, cipher_calculated, plain_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl,
                                   cipher_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   plain_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-GCM Example

This is an example of AES-GCM encryption/decryption.

```
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_NONCE_LEN (12)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (8)
void r_rsip_example_gcm ()
{
 fsp_err_t            err;
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
```

```
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                = {0};
    uint8_t tag[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                   = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
```

```
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) */
    err            = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
    input_length = 0;
    output_length = 0;
 /* Input ciphertext (multi-shot) */
    input_length_tmp = 10;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
```

```
    input_length_tmp = GCM_TEXT_LEN - input_length;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /*
  * Output remaining text and verify the tag
  * Attention: Do not use the output plaintext by R_RSIP_AES_Cipher_DecryptUpdate()
  * before R_RSIP_AES_Cipher_DecryptFinal() returns FSP_SUCCESS.
  * The integrity is checked in this function.
  */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-GMAC Example

This is an example of AES-GMAC signature/verification.

```
#define GMAC_NONCE_LEN (12)
#define GMAC_AAD_LEN (8)
#define GMAC_TAG_LEN (16)
void r_rsip_example_gmac ()
{
 fsp_err_t err;
 static const uint8_t aad[GMAC_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
```

```
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GMAC_NONCE_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t tag[GMAC_TAG_LEN] BSP_ALIGN_VARIABLE(4)    = {0};
    uint8_t output_dummy_buf[1] = {0};
    uint32_t output_length       = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Initialize GMAC signature */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC generation */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, output_dummy_buf, &output_length,
tag);
```

```
    assert(FSP_SUCCESS == err);
 /*Initialize GMAC verification*/
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC verification*/
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, output_dummy_buf, &output_length, tag,
GMAC_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CCM Example

This is an example of AES-CCM encryption/decryption.

```
#define CCM_NONCE_LEN (12)
#define CCM_ADATA_LEN (20)
#define CCM_INPUT_DATA_LEN (24)
#define CCM_MAC_LEN (8)
void r_rsip_example_ccm ()
{
 fsp_err_t err;
 static const uint8_t nonce[CCM_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B};
 static const uint8_t adata[CCM_ADATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
       0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13
    };
```

```
 static const uint8_t plain[CCM_INPUT_DATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29,
       0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33,
       0x34, 0x35, 0x36, 0x37
    };
    uint8_t   cipher_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   plain_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   mac[CCM_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length_tmp = 0;
    uint8_t * p_output          = NULL;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_ENC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
```

```
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(cipher_calculated[output_length_tmp]);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, p_output, &output_length_tmp, mac);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_DEC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
 // err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
plain_calculated, &output_length_tmp);
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                 cipher_calculated,
                                 CCM_INPUT_DATA_LEN,
                                 plain_calculated,
                                 &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(plain_calculated[output_length_tmp]);
 /* Output remaining text and verify the tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, p_output, &output_length_tmp, mac,
CCM_MAC_LEN);
```

```
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CMAC Example

This is an example of AES-CMAC signature/verification.

```
#define CMAC_MESSAGE_LEN (10)
#define CMAC_MAC_LEN (16)
void r_rsip_example_cmac ()
{
 fsp_err_t err;
 static const uint8_t message[CMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0xf0, 0x8f, 0x89, 0x08, 0x75, 0xe1, 0x39, 0x48, 0x04, 0x89
    };
    uint8_t mac[CMAC_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for encrypt */
```

```
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_MAC_SignFinish(&g_rsip_ctrl, mac);
    assert(FSP_SUCCESS == err);
 /* CMAC initialize for decrypt */
    err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Input message */
    err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Verify the tag */
    err = R_RSIP_AES_MAC_VerifyFinish(&g_rsip_ctrl, mac, CMAC_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ECC Example

This is an example of ECDSA signature generation/verification.

```
#define SECP256R1_KEY_SIZE (32)
void r_rsip_example_ecc ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
    uint8_t hash[SECP256R1_KEY_SIZE] BSP_ALIGN_VARIABLE(4)          = {0};
    uint8_t signature[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
```

```
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_ECDSA_Verify(&g_rsip_ctrl, &wrapped_public_key, hash, signature);
    assert(FSP_SUCCESS == err);
```

```
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### RSA Example

This is an example of RSA encryption/decryption (RSAES-OAEP) and RSA signature generation/verification (RSASSA-PSS).

```
#define SHA256_HASH_SIZE (32)
#define RSA_2048_KEY_SIZE (256)
void r_rsip_example_rsa ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
 static const uint8_t label[] BSP_ALIGN_VARIABLE(4) = "label";
    uint8_t cipher_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t plain_calculated_length = 0;
    uint8_t hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)       = {0};
    uint8_t signature[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_RSA_2048_PUBLIC, .p_value = &wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
```

```
      .type = RSIP_KEY_TYPE_RSA_2048_PRIVATE, .p_value = &wrapped_private_key_value
   };
/* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* Generate a key pair of RSA-2048 */
   err =
R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key, &wrapped_private_key);
   assert(FSP_SUCCESS == err);
/*
 * RSAES-OAEP
 */
/* Encrypt a plaintext */
   err = R_RSIP_RSAES_OAEP_Encrypt(&g_rsip_ctrl,
                                   &wrapped_public_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                   message,
sizeof(message),
                                   cipher_calculated);
   assert(FSP_SUCCESS == err);
/* Decrypt a ciphertext using the same key */
   err = R_RSIP_RSAES_OAEP_Decrypt(&g_rsip_ctrl,
                                   &wrapped_private_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                  cipher_calculated,
                                  plain_calculated,
                                   &plain_calculated_length,
sizeof(plain_calculated));
```

```
    assert(FSP_SUCCESS == err);
 /* Verify calculated plaintext */
    assert(sizeof(message) == plain_calculated_length);
    assert(0 == memcmp(message, plain_calculated, plain_calculated_length));
 /*
  * RSASSA-PSS
  */
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_RSASSA_PSS_Sign(&g_rsip_ctrl,
                                 &wrapped_private_key,
 RSIP_HASH_TYPE_SHA256,
 RSIP_MGF_TYPE_MGF1_SHA256,
 RSIP_RSA_SALT_LENGTH_AUTO,
                                 hash,
                                 signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_RSASSA_PSS_Verify(&g_rsip_ctrl,
                                  &wrapped_public_key,
 RSIP_HASH_TYPE_SHA256,
 RSIP_MGF_TYPE_MGF1_SHA256,
 RSIP_RSA_SALT_LENGTH_AUTO,
                                   hash,
                                   signature);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HASH Example

This is an example of calculating the SHA-256 message digest.

```
#define SHA256_HASH_SIZE (32)
void r_rsip_example_hash ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE (4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
       0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    };
 static const uint8_t hash[] BSP_ALIGN_VARIABLE (4) =
    {
       0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
       0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
    };
    uint8_t digest_single[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t digest_multi[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate SHA-256 digests with single-part API */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), digest_single);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_single, hash, SHA256_HASH_SIZE));
 /* (2) Calculate SHA-256 digests with multi-part API */
    err = R_RSIP_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
```

```
    err = R_RSIP_SHA_Update(&g_rsip_ctrl, message, sizeof(message));
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Finish(&g_rsip_ctrl, digest_multi);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_multi, hash, SHA256_HASH_SIZE));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HMAC Example

This is an example of HMAC signature/verification.

```
#define HMAC_MESSAGE_LEN (34)
#define HMAC_HASH_LEN (32)
void r_rsip_example_hmac ()
{
 fsp_err_t err;
 static const uint8_t message[HMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x53, 0x61, 0x6D, 0x70, 0x6C, 0x65, 0x20, 0x6D,
       0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x20, 0x66,
       0x6F, 0x72, 0x20, 0x6B, 0x65, 0x79, 0x6C, 0x65,
       0x6E, 0x3C, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x6C,
       0x65, 0x6E
    };
    uint8_t hash_1[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t hash_2[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
```

```
    {
        .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a HMAC-SHA 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to finish */
    err = R_RSIP_HMAC_Compute(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1);
    assert(FSP_SUCCESS == err);
 /* (2) Verify HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to verify */
    err = R_RSIP_HMAC_Verify(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* (3) Calculate HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC generation */
    err = R_RSIP_HMAC_SignFinish(&g_rsip_ctrl, hash_2);
    assert(FSP_SUCCESS == err);
 /* (4) Verify HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
```

```
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC verification */
    err = R_RSIP_HMAC_VerifyFinish(&g_rsip_ctrl, hash_2, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare hash_1 and hash_2 */
    assert(0 == memcmp(hash_1, hash_2, HMAC_HASH_LEN));
}
```

### PKI Example

This is an example that verifies the public key certificate, imports the peer's public key, and calculates the shared secret.

```
#define SHA256_HASH_SIZE (32)
#define SECP256R1_KEY_SIZE (32)
/* Simplified public key certificate structure */
typedef struct st_cert
{
    uint8_t public_key_param_qx[SECP256R1_KEY_SIZE];
    uint8_t public_key_param_qy[SECP256R1_KEY_SIZE];
} cert_t;
void r_rsip_example_pki ()
{
 fsp_err_t err;
 /* Prepare data area for generating certificate and shared secret */
    cert_t               cert;
    uint8_t              raw_public_key[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t              hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t              signature[SECP256R1_KEY_SIZE * 2]
```

```
BSP_ALIGN_VARIABLE(4)      = {0};
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare wrapped key data area for CA (Virtually prepared for local validation) */
    uint8_t
wrapped_ca_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_ca_public_key_value
    };
    uint8_t
wrapped_ca_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_ca_private_key_value
    };
 /* Prepare wrapped key data area for peer (Virtually prepared for local validation)
*/
    uint8_t
wrapped_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_peer_public_key_value
    };
    uint8_t
wrapped_peer_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
```

```
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_peer_private_key_value
    };
 /* Prepare wrapped key data area for Local device */
    uint8_t
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 /* Prepare wrapped key data area for verified peer public key */
    uint8_t wrapped_verified_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
 RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_verified_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_verified_peer_public_key_value
    };
```

```
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate key pairs for CA, peer, and Local device */
 /* Generate a key pair of secp256r1 for CA (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_ca_public_key,
&wrapped_ca_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for peer (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_peer_public_key,
&wrapped_peer_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device (Assume product with RSIP
driver) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
/* CA process :
 * In the following, generate certificate and signature of peer public key */
 /* Export raw peer public key (Qx|Qy) and store it in cert */
    err = R_RSIP_PublicKeyExport(&wrapped_peer_public_key, raw_public_key);
    assert(FSP_SUCCESS == err);
    memcpy(cert.public_key_param_qx, &raw_public_key[0], SECP256R1_KEY_SIZE);
    memcpy(cert.public_key_param_qy, &raw_public_key[SECP256R1_KEY_SIZE],
SECP256R1_KEY_SIZE);
 /* Generate the hash of cert and store it in hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, (uint8_t *) &cert,
sizeof(cert), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature of hash with CA private key and store it in signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_ca_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
```

```
/* Local device process :
 * In the following, verify the CA certificate, import peer public key, and generate
shared secret */
 /* Verify the signature of cert with CA public key */
    err = R_RSIP_PKI_ECDSA_CertVerify(&g_rsip_ctrl, &wrapped_ca_public_key, hash,
signature);
    assert(FSP_SUCCESS == err);
 /* Import verified peer public key with wrapped format */
    err = R_RSIP_PKI_CertKeyImport(&g_rsip_ctrl,
                                   (uint8_t *) &cert,
 sizeof(cert),
                                   cert.public_key_param_qx,
                                   SECP256R1_KEY_SIZE,
                                   cert.public_key_param_qy,
                                   SECP256R1_KEY_SIZE,
 RSIP_HASH_TYPE_SHA256,
                                   &wrapped_verified_peer_public_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the Local private key and the verified peer public
key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl,
                               &wrapped_local_private_key,
                               &wrapped_verified_peer_public_key,
                               &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### KDF-SHA Example

This is an example of One-Step Key Derivation (NIST SP 800-56C) using KDF-SHA.

```
#define COUNTER_BYTE_LEN (4)
```

```
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (64)
#define CONCAT_DKM_BYTE_SIZE KDF_SHA256_DKM_BYTE_SIZE * 2
#define WRAPPED_IV_BYTE_SIZE (36)
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (16)
void r_rsip_example_kdf_sha ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_DKM_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_SHA256_DKM_BYTE_SIZE];
```

```
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-SHA operation */
    uint32_t             counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
```

```
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* One-Step Key Derivation (NIST SP 800-56C)
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter = 0x00000000;
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
```

```
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM1 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM2 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_DKM_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In One-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
```

```
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   16,
                                   NULL,
                                   wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
```

```
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, cipher_calculated, GCM_TEXT_LEN,
plain_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### KDF-HMAC Example

This is an example of Two-Step Key Derivation (NIST SP 800-56C) using KDF-HMAC.

```
#define INITIAL_VALUE_BYTE_LEN (32)
#define FIXED_INFO_BYTE_LEN (16)
#define HMAC_SHA256_KEY_LEN (32)
#define KDF_HMAC_SHA256_MAC_BYTE_SIZE (64)
#define CONCAT_MAC_BYTE_SIZE KDF_HMAC_SHA256_MAC_BYTE_SIZE * 2
#define WRAPPED_IV_BYTE_SIZE (36)
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_AAD_LEN (8)
```

```
#define GCM_TAG_LEN (16)
void r_rsip_example_kdf_hmac ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped key as a salt */

uint8_t            wrapped_salt_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_salt =
    {
       .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_salt_value
    };
 /* Prepare data area for wrapped KDK (Key Derivation Key) */
```

```
uint8_t            wrapped_kdk_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_kdk =
    {
       .type = RSIP_KEY_TYPE_KDF_HMAC_SHA256, .p_value = &wrapped_kdk_value
    };
 /* Prepare data area for wrapped MAC (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_MAC_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_HMAC_SHA256_MAC_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the MAC */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-HMAC operation */
 static const uint8_t initial_value[INITIAL_VALUE_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
    };
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
```

```
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 1 : Randomness Extraction
 * Generate KDK (Key Derivation Key) */
/* In the following, calculate KDK = KDF-HMAC[salt, Z], where Z is shared secret */
```

```
 /* Generate a HMAC-SHA 256 key as a salt */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Initialize KDF-HMAC operation (Key setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_HMAC_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Convert the wrapped MAC to wrapped HMAC-SHA256 key, which is called KDK (Key
Derivation Key) */
    err = R_RSIP_KDF_HMAC_DKMKeyImport(&g_rsip_ctrl, p_wrapped_dkm1,
HMAC_SHA256_KEY_LEN, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 2 : Key Expansion
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-HMAC[KDK, K(0)|FixedInfo], where K(0) = IV
*/
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
 /* Input K(0) = IV as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, initial_value,
INITIAL_VALUE_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
```

```
/* In the following, calculate K(2) = KDF-HMAC[KDK, K(1)|FixedInfo] */
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
 /* Input K(1) as a ciphertext */
    err = R_RSIP_KDF_HMAC_DKMUpdate(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_MAC_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In Two-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   16,
                                   NULL,
```

```
                                    wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, cipher_calculated, GCM_TEXT_LEN,
plain_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
```

```
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### 5.2.15.7 Renesas Secure IP (r_rsip_protected)

Modules » Security

#### Functions

| | |
|---|---|
| fsp_err_t | R_RSIP_Open (rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg) |
| fsp_err_t | R_RSIP_Close (rsip_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RSIP_RandomNumberGenerate (rsip_ctrl_t *const p_ctrl, uint8_t *const p_random) |
| fsp_err_t | R_RSIP_KeyGenerate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_KeyPairGenerate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key) |
| fsp_err_t | R_RSIP_EncryptedKeyWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_RFC3394_KeyWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key) |
| fsp_err_t | R_RSIP_RFC3394_KeyUnwrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const |

| | |
|---|---|
| | *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key) |
| fsp_err_t | R_RSIP_PublicKeyExport (rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t *const p_raw_public_key) |
| fsp_err_t | R_RSIP_AES_Cipher_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector) |
| fsp_err_t | R_RSIP_AES_Cipher_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length) |
| fsp_err_t | R_RSIP_AES_Cipher_Finish (rsip_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RSIP_AES_AEAD_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length) |
| fsp_err_t | R_RSIP_AES_AEAD_LengthsSet (rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length) |
| fsp_err_t | R_RSIP_AES_AEAD_AADUpdate (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length) |
| fsp_err_t | R_RSIP_AES_AEAD_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| fsp_err_t | R_RSIP_AES_AEAD_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag) |
| fsp_err_t | R_RSIP_AES_AEAD_Verify (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length) |
| fsp_err_t | R_RSIP_AES_MAC_Init (rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key) |
| fsp_err_t | R_RSIP_AES_MAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| fsp_err_t | R_RSIP_AES_MAC_SignFinish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac) |
| fsp_err_t | R_RSIP_AES_MAC_VerifyFinish (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length) |
| fsp_err_t | R_RSIP_ChaCha20_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t |

| | |
|---|---|
| | const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter) |
| fsp_err_t | R_RSIP_ChaCha20_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| fsp_err_t | R_RSIP_ChaCha20_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length) |
| fsp_err_t | R_RSIP_ChaCha20_Poly1305_Init (rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length) |
| fsp_err_t | R_RSIP_ChaCha20_Poly1305_AADUpdate (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length) |
| fsp_err_t | R_RSIP_ChaCha20_Poly1305_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| fsp_err_t | R_RSIP_ChaCha20_Poly1305_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag) |
| fsp_err_t | R_RSIP_ChaCha20_Poly1305_Verify (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length) |
| fsp_err_t | R_RSIP_ECDSA_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature) |
| fsp_err_t | R_RSIP_ECDSA_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_PKI_ECDSA_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_PureEdDSA_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature) |
| fsp_err_t | R_RSIP_PureEdDSA_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature) |

| | |
|---|---|
| fsp_err_t | R_RSIP_ECDH_KeyAgree (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_t *const p_wrapped_secret) |
| fsp_err_t | R_RSIP_ECDH_PlainKeyAgree (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_t *const p_wrapped_secret) |
| fsp_err_t | R_RSIP_KDF_SHA_Init (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type) |
| fsp_err_t | R_RSIP_KDF_SHA_ECDHSecretUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret) |
| fsp_err_t | R_RSIP_KDF_SHA_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| fsp_err_t | R_RSIP_KDF_SHA_Finish (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm) |
| fsp_err_t | R_RSIP_KDF_SHA_Suspend (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_t *const p_handle) |
| fsp_err_t | R_RSIP_KDF_SHA_Resume (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_t const *const p_handle) |
| fsp_err_t | R_RSIP_KDF_HMAC_DKMKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_KDF_HMAC_ECDHSecretKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_KDF_HMAC_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key) |
| fsp_err_t | R_RSIP_KDF_HMAC_DKMUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm) |
| fsp_err_t | R_RSIP_KDF_HMAC_ECDHSecretUpdate (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_t const *const p_wrapped_secret) |
| fsp_err_t | R_RSIP_KDF_HMAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| fsp_err_t | R_RSIP_KDF_HMAC_SignFinish (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm) |

| | |
|---|---|
| fsp_err_t | R_RSIP_KDF_HMAC_Suspend (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_t *const p_handle) |
| fsp_err_t | R_RSIP_KDF_HMAC_Resume (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_t const *const p_handle) |
| fsp_err_t | R_RSIP_KDF_DKMConcatenate (rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length) |
| fsp_err_t | R_RSIP_KDF_DerivedKeyImport (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_KDF_DerivedIVWrap (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector) |
| fsp_err_t | R_RSIP_OTF_Init (rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed) |
| fsp_err_t | R_RSIP_PKI_VerifiedCertInfoExport (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_t *const p_verified_cert_info) |
| fsp_err_t | R_RSIP_PKI_VerifiedCertInfoImport (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_t const *const p_verified_cert_info) |
| fsp_err_t | R_RSIP_PKI_CertKeyImport (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key) |
| fsp_err_t | R_RSIP_RSA_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher) |
| fsp_err_t | R_RSIP_RSA_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain) |
| fsp_err_t | R_RSIP_RSAES_PKCS1_V1_5_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher) |
| fsp_err_t | R_RSIP_RSAES_PKCS1_V1_5_Decrypt (rsip_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length) |
| fsp_err_t | R_RSIP_RSAES_OAEP_Encrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher) |
| fsp_err_t | R_RSIP_RSAES_OAEP_Decrypt (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length) |
| fsp_err_t | R_RSIP_RSASSA_PKCS1_V1_5_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature) |
| fsp_err_t | R_RSIP_RSASSA_PKCS1_V1_5_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_RSASSA_PSS_Sign (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature) |
| fsp_err_t | R_RSIP_RSASSA_PSS_Verify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_PKI_RSASSA_PSS_CertVerify (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| fsp_err_t | R_RSIP_SHA_Compute (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t |

| | |
|---|---|
| | const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest) |
| fsp_err_t | R_RSIP_SHA_Init (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type) |
| fsp_err_t | R_RSIP_SHA_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| fsp_err_t | R_RSIP_SHA_Finish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest) |
| fsp_err_t | R_RSIP_SHA_Suspend (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_t *const p_handle) |
| fsp_err_t | R_RSIP_SHA_Resume (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_t const *const p_handle) |
| fsp_err_t | R_RSIP_HMAC_Compute (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac) |
| fsp_err_t | R_RSIP_HMAC_Verify (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length) |
| fsp_err_t | R_RSIP_HMAC_Init (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_Update (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| fsp_err_t | R_RSIP_HMAC_SignFinish (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac) |
| fsp_err_t | R_RSIP_HMAC_VerifyFinish (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length) |
| fsp_err_t | R_RSIP_HMAC_Suspend (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_t *const p_handle) |
| fsp_err_t | R_RSIP_HMAC_Resume (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_t const *const p_handle) |
| fsp_err_t | R_RSIP_FSBL_OEM_BL_Digest_Generate (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_key_cert, uint32_t const key_cert_max_length, uint8_t const *const p_code_cert, uint32_t const code_cert_max_length, uint32_t *const p_mac) |

## Detailed Description

Driver for the Renesas Secure IP on RA MPUs. This module implements the RSIP Interface.

# Overview

This module provides RSIP functions in protected mode.

### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E11A | RA4L1 |
| RSIP-E31A | RA4C1 |
| RSIP-E51A | RA8M1, RA8D1, RA8T1 |
| RSIP-E50D | RA8M2, RA8D2, RA8T2, RA8P1 |

### Features

The RSIP module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/192/256bit
      - CBC 128/192/256bit
      - CTR 128/192/256bit
      - GCM 128/192/256bit
      - CCM 128/192/256bit
      - XTS 128/256bit
    - ChaCha
      - ChaCha20
      - ChaCha20-Poly1305
  - Asymmetric Encryption/Decryption
    - RSA
      - RSAES-PKCS1-V1_5 2048/3072/4096bit
      - RSAES-OAEP 2048/3072/4096bit
      - RSASSA-PKCS1-V1_5 2048/3072/4096bit
      - RSASSA-PSS 2048/3072/4096bit
    - ECC
      - ECDSA
        - secp256r1/secp384r1/secp521r1
        - secp256k1
        - brainpool P256r1/P384r1/P512r1
      - EdDSA
        - Ed25519
  - Key Agreement
    - ECDH
      - secp256r1/secp384r1/secp521r1
      - brainpool P256r1/P384r1
  - Key Derivation
    - Key Derivation Function (KDF) using SHA
      - SHA-256/384bit
    - Key Derivation Function (KDF) using HMAC
      - HMAC-SHA256/384/512bit

        - Key Import by derived keying material
        - IV output by derived keying material
    - Public Key Infrastructure (PKI)
        - Certificate Verification
            - ECDSA
                - secp256r1/secp384r1/secp521r1
                - brainpool P256r1/P384r1
        - Key Import from Certificate
            - ECC
                - secp256r1/secp384r1/secp521r1
                - brainpool P256r1/P384r1
    - Hash Functions
        - SHA-2
            - SHA-224, 256, 384, 512, 512/224, 512/256
        - SHA-3
            - SHA3-224, 256, 384, 512
    - RFC3394 Key Wrap/Unwrap
- Message Authentication Code
    - HMAC-SHA224, 256, 384, 512, 512/224, 512/256 bit
    - AES-CMAC 128/192/256bit
- Key Manage
    - Key Support
        - AES 128/192/256bit
        - RSA 2048/3072/4096bit
        - ECC
            - secp256r1/secp384r1/secp521r1
            - secp256k1
            - brainpool P256r1/P384r1/P512r1
        - HMAC-SHA224, 256, 384, 512, 512/224, 512/256 bit
- Random Number Generate
- Secure boot for FSBL
    - Manifest verification and MAC generation (Refer to Application Note R11AN0774.)

**Supported algorithms**

The following algorithms are available on each devices.

- RSIP-E11A: RA4L1
- RSIP-E31A: RA4C1
- RSIP-E51A: RA8M1, RA8D1, RA8T1
- RSIP-E50D: RA8M2, RA8D2, RA8T2, RA8P1

**Symmetric Key**

| Key type (rsip_key_type_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_KEY_TYPE_AES_128 | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_AES_192 | Not Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_AES_256 | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_XT | Not Supported | Not Supported | Supported | Supported |

| | | | | |
|---|---|---|---|---|
| S_AES_128 | | | | |
| RSIP_KEY_TYPE_XTS_AES_256 | Not Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_CHACHA20 | Not Supported | Not Supported | Not Supported | Supported |
| RSIP_KEY_TYPE_HMAC_SHA224 | Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_HMAC_SHA256 | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_HMAC_SHA384 | Not Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_HMAC_SHA512 | Not Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_HMAC_SHA512_224 | Not Supported | Supported | Not Supported | Not Supported |
| RSIP_KEY_TYPE_HMAC_SHA512_256 | Not Supported | Supported | Not Supported | Not Supported |

**Asymmetric Key**

| Public key type ( rsip_key_type_t) | Private key type ( rsip_key_type_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|---|
| RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE | Not Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE | Not Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC | RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE | Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC | RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE | Not Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE | RSIP_KEY_TYPE | Not Supported | Not Supported | Supported | Supported |

| | | | | | |
|---|---|---|---|---|---|
| _ECC_BRAINPOOLP512R1_PUBLIC | _ECC_BRAINPOOLP512R1_PRIVATE | | | | |
| RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC | RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE | Not Supported | Supported | Supported | Supported |
| RSIP_KEY_TYPE_RSA_2048_PUBLIC | RSIP_KEY_TYPE_RSA_2048_PRIVATE | Not Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_RSA_3072_PUBLIC | RSIP_KEY_TYPE_RSA_3072_PRIVATE | Not Supported | Not Supported | Supported | Supported |
| RSIP_KEY_TYPE_RSA_4096_PUBLIC | RSIP_KEY_TYPE_RSA_4096_PRIVATE | Not Supported | Not Supported | Supported | Supported |

### AES Block Cipher Mode of Operation

| Cipher mode (rsip_aes_cipher_mode_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_AES_CIPHER_MODE_ECB_ENC | Supported | Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_ECB_DEC | Supported | Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_CBC_ENC | Supported | Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_CBC_DEC | Supported | Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_CTR | Supported | Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_XTS_ENC | Not Supported | Not Supported | Supported | Supported |
| RSIP_AES_CIPHER_MODE_XTS_DEC | Not Supported | Not Supported | Supported | Supported |

| AEAD mode (rsip_aes_aead_mode_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_AES_AEAD_MODE_GCM_ENC | Supported | Supported | Supported | Supported |
| RSIP_AES_AEAD_MODE_GCM_DEC | Supported | Supported | Supported | Supported |
| RSIP_AES_AEAD_M | Supported | Supported | Supported | Supported |

| | | | | |
|---|---|---|---|---|
| ODE_CCM_ENC | | | | |
| RSIP_AES_AEAD_MODE_CCM_DEC | Supported | Supported | Supported | Supported |

| MAC mode (rsip_aes_mac_mode_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_AES_MAC_MODE_CMAC | Supported | Supported | Supported | Supported |

**ChaCha20-Poly1305 Operation**

This feature has a usage constraint that requires an input of at least one byte of encrypt/decrypt text data.

| ChaCha20-Poly1305 mode (rsip_chacha_poly_mode_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_CHACHA20_POLY1305_MODE_ENC | Not Supported | Not Supported | Not Supported | Supported |
| RSIP_CHACHA20_POLY1305_MODE_DEC | Not Supported | Not Supported | Not Supported | Supported |

**Hash Function**

Using hardware-accelerated SHA-3 in Protected Mode for use with a software PQC library is prohibited by the Protected Mode security policy due to the requirement for plaintext key usage with a software cryptographic library.

| Hash Function (rsip_hash_type_t) | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| RSIP_HASH_TYPE_SHA224 | Supported | Supported | Supported | Supported |
| RSIP_HASH_TYPE_SHA256 | Supported | Supported | Supported | Supported |
| RSIP_HASH_TYPE_SHA384 | Not Supported | Supported | Supported | Supported |
| RSIP_HASH_TYPE_SHA512 | Not Supported | Supported | Supported | Supported |
| RSIP_HASH_TYPE_SHA512_224 | Not Supported | Supported | Supported | Supported |
| RSIP_HASH_TYPE_SHA512_256 | Not Supported | Supported | Supported | Supported |

| | | | | |
|---|---|---|---|---|
| RSIP_HASH_TYPE_SHA3_224 | Not Supported | Not Supported | Not Supported | Supported |
| RSIP_HASH_TYPE_SHA3_256 | Not Supported | Not Supported | Not Supported | Supported |
| RSIP_HASH_TYPE_SHA3_384 | Not Supported | Not Supported | Not Supported | Supported |
| RSIP_HASH_TYPE_SHA3_512 | Not Supported | Not Supported | Not Supported | Supported |

**Key Derivation Function**

| Key Derivation Function | RSIP-E11A | RSIP-E31A | RSIP-E51A | RSIP-E50D |
|---|---|---|---|---|
| SHA-256 | Supported | Supported | Supported | Supported |
| SHA-384 | Not Supported | Supported | Supported | Supported |
| HMAC-SHA256 | Not Supported | Supported | Supported | Supported |
| HMAC-SHA384 | Not Supported | Supported | Supported | Supported |
| HMAC-SHA512 | Not Supported | Not Supported | Supported | Supported |

# Configuration

### Build Time Configurations for r_rsip_e11a_protected

The following build time configurations are defined in fsp_cfg/r_rsip_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-192 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the |

| | | | |
|---|---|---|---|
| | | | functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-128 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused |

| | | | |
|---|---|---|---|
| | | | algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20-Poly1305 | Disabled | Disabled | If the application uses only some of the algorithm in the |

| | | | |
|---|---|---|---|
| | | | function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP384R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |

| | | | |
|---|---|---|---|
| ECC BRAINPOOLP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP384R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC EDWARDS25519 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| RSA-3072 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-4096 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-224 | Disabled | Disabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it |

| | | | |
|---|---|---|---|
| | | | works effectively. |
| HMAC_SHA224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This |

| | | | |
|---|---|---|---|
| | | | configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Build Time Configurations for r_rsip_e31a_protected**

The following build time configurations are defined in fsp_cfg/r_rsip_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-192 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size |

| | | | |
|---|---|---|---|
| | | | can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-128 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled | Enabled | If the application uses |

| | | | |
|---|---|---|---|
| | • Disabled | | only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20-Poly1305 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the |

| | | | |
|---|---|---|---|
| | | | functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused |

| | | | |
|---|---|---|---|
| | | | algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC EDWARDS25519 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-3072 | Disabled | Disabled | If the application uses only some of the algorithm in the |

| | | | |
|---|---|---|---|
| | | | function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-4096 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |

| | | | |
|---|---|---|---|
| SHA-384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| SHA3-256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-384 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-512 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Build Time Configurations for r_rsip_e51a_protected**

The following build time configurations are defined in fsp_cfg/r_rsip_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-192 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |

| | | | |
|---|---|---|---|
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20-Poly1305 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC EDWARDS25519 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-3072 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it |

| | | | |
|---|---|---|---|
| | | | works effectively. |
| RSA-4096 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This |

| | | | |
|---|---|---|---|
| | | | configuration is provided only for the functions for which it works effectively. |
| SHA-512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-224 | Disabled | config.driver.rsip.algorithms_enable.sha3_224.enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-256 | Disabled | config.driver.rsip.algorithms_enable.sha3_256.enabled | If the application uses only some of the algorithm in the function, the code size |

| | | | |
|---|---|---|---|
| | | | can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-384 | Disabled | config.driver.rsip.algorithms_enable.sha3_384.enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-512 | Disabled | config.driver.rsip.algorithms_enable.sha3_512.enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled | Enabled | If the application uses |

| | | | |
|---|---|---|---|
| | • Disabled | | only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the |

| | | | |
|---|---|---|---|
| | | | functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Build Time Configurations for r_rsip_e50d_protected**

The following build time configurations are defined in fsp_cfg/r_rsip_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Selection of algorithms to enable | | | |
| AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-192 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-ECB CBC CTR | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused |

| | | | |
|---|---|---|---|
| | | | algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-128 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| XTS-AES | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-GCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CCM | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the |

| | | | |
|---|---|---|---|
| | | | function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| AES-CMAC | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ChaCha20-Poly1305 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |

| | | | |
|---|---|---|---|
| ECC SECP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP521R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC SECP256K1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP256R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP384R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is |

| | | | |
|---|---|---|---|
| | | | provided only for the functions for which it works effectively. |
| ECC BRAINPOOLP512R1 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| ECC EDWARDS25519 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-2048 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-3072 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| RSA-4096 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by |

| | | | |
|---|---|---|---|
| | | | disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the |

| | | | |
|---|---|---|---|
| | | | algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA-512_256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it |

| | | | |
|---|---|---|---|
| | | | works effectively. |
| SHA3-384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| SHA3-512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA1 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA224 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA256 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This |

| | | | |
|---|---|---|---|
| | | | configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA384 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512 | • Enabled<br>• Disabled | Enabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_224 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| HMAC_SHA512_256 | Disabled | Disabled | If the application uses only some of the algorithm in the function, the code size can be reduced by disabling the unused algorithms. This configuration is provided only for the functions for which it works effectively. |
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Getting Started: Creating a RSIP Protected Mode Project

Start by creating a new project in e² studio or RASC. On the Stacks tab, add New > Security > RSIP Protected Mode (r_rsip_protected).

### State Transition

This driver has the following states.

| State Name | Details |
|---|---|
| STATE_INITIAL | Driver is not open. |
| STATE_MAIN | Ready to Compute. |
| STATE_AES | Computing AES (unauthenticated cipher) |
| STATE_AES_AEAD | Computing AES (AEAD). |
| STATE_AES_MAC | Computing AES (MAC). |
| STATE_CHACHA | Computing ChaCha20. |
| STATE_CHACHA_POLY | Computing ChaCha20-Poly1305. |
| STATE_SHA | Computing SHA. |
| STATE_HMAC | Computing HMAC. |
| STATE_KDF_SHA | Computing KDF SHA. |
| STATE_KDF_HMAC | Computing KDF HMAC. |

There are two types of APIs provided by the RSIP driver for accelerating cryptographic operations: those that provide cryptographic operations in a single API and those that provide them in multiple APIs. In this document, the former is referred to as single-part operations and the latter as multi-part operations. Each corresponds to the following algorithms:

- Single-part operations: DRBG, Key management, ECC, RSA, SHA, HMAC
- Multi-part operations: AES, SHA, HMAC

Multi-part operations are APIs which split a single cryptographic operation into a sequence of separate steps (e.g. Init-Update-Finish). This enables fine control over the configuration of the cryptographic operation, and allows the message data to be processed in fragments instead of all at once.

Due to the above characteristics, it is necessary to manage the operational states in the multi-part operations.

# Examples

### AES cipher Example

This is an example of AES-CBC encryption/decryption.

```
void r_rsip_example_aes ()
{
 fsp_err_t            err;
 static const uint8_t plain[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
    };
    uint8_t iv[RSIP_PRV_BYTE_SIZE_AES_BLOCK] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t plain_calculated[RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2] BSP_ALIGN_VARIABLE(4)
= {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
```

```
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, iv);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, plain, cipher_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_Cipher_Init(&g_rsip_ctrl, RSIP_AES_CIPHER_MODE_CBC_DEC,
&wrapped_key, iv);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl, cipher_calculated, plain_calculated,
RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Update(&g_rsip_ctrl,
                                   cipher_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   plain_calculated + RSIP_PRV_BYTE_SIZE_AES_BLOCK,
                                   RSIP_PRV_BYTE_SIZE_AES_BLOCK);
    assert(FSP_SUCCESS == err);
    err = R_RSIP_AES_Cipher_Finish(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, RSIP_PRV_BYTE_SIZE_AES_BLOCK * 2));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-GCM Example

This is an example of AES-GCM encryption/decryption.

```
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_NONCE_LEN (12)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (8)
void r_rsip_example_gcm ()
{
 fsp_err_t            err;
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                = {0};
    uint8_t tag[GCM_BLOCK_LEN] BSP_ALIGN_VARIABLE(4)                   = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
```

```
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) */
    err            = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
    input_length = 0;
```

```
    output_length = 0;
 /* Input ciphertext (multi-shot) */
    input_length_tmp = 10;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
    input_length_tmp = GCM_TEXT_LEN - input_length;
    err             = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    output_length += output_length_tmp;
    assert(FSP_SUCCESS == err);
 /*
  * Output remaining text and verify the tag
  * Attention: Do not use the output plaintext by R_RSIP_AES_Cipher_DecryptUpdate()
  * before R_RSIP_AES_Cipher_DecryptFinal() returns FSP_SUCCESS.
  * The integrity is checked in this function.
  */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-GMAC Example

This is an example of AES-GMAC signature/verification.

```
#define GMAC_NONCE_LEN (12)
#define GMAC_AAD_LEN (8)
#define GMAC_TAG_LEN (16)
void r_rsip_example_gmac ()
{
 fsp_err_t err;
 static const uint8_t aad[GMAC_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
    };
    uint8_t nonce[GMAC_NONCE_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t tag[GMAC_TAG_LEN] BSP_ALIGN_VARIABLE(4)    = {0};
    uint8_t output_dummy_buf[1] = {0};
    uint32_t output_length       = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Generate a nonce */
    err = R_RSIP_RandomNumberGenerate(&g_rsip_ctrl, nonce);
```

```
    assert(FSP_SUCCESS == err);
 /* Initialize GMAC signature */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC generation */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, output_dummy_buf, &output_length,
tag);
    assert(FSP_SUCCESS == err);
 /*Initialize GMAC verification*/
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC,
&wrapped_key, nonce, GMAC_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GMAC_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize GMAC verification*/
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, output_dummy_buf, &output_length, tag,
GMAC_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CCM Example

This is an example of AES-CCM encryption/decryption.

```
#define CCM_NONCE_LEN (12)
#define CCM_ADATA_LEN (20)
#define CCM_INPUT_DATA_LEN (24)
```

```
#define CCM_MAC_LEN (8)
void r_rsip_example_ccm ()
{
 fsp_err_t err;
 static const uint8_t nonce[CCM_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B};
 static const uint8_t adata[CCM_ADATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
       0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13
    };
 static const uint8_t plain[CCM_INPUT_DATA_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29,
       0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33,
       0x34, 0x35, 0x36, 0x37
    };
    uint8_t   cipher_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   plain_calculated[CCM_INPUT_DATA_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t   mac[CCM_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length_tmp = 0;
    uint8_t * p_output          = NULL;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a aes 256-bit key */
```

```
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Encrypt a plaintext */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_ENC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(cipher_calculated[output_length_tmp]);
 /* Output remaining text and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, p_output, &output_length_tmp, mac);
    assert(FSP_SUCCESS == err);
 /* Decrypt a ciphertext using the same key */
    err = R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_CCM_DEC,
&wrapped_key, nonce, CCM_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Set text and tag length. */
    err = R_RSIP_AES_AEAD_LengthsSet(&g_rsip_ctrl, CCM_ADATA_LEN, CCM_INPUT_DATA_LEN,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, adata, CCM_ADATA_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext */
 // err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, CCM_INPUT_DATA_LEN,
```

```
plain_calculated, &output_length_tmp);
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl,
                                 cipher_calculated,
                                 CCM_INPUT_DATA_LEN,
                                 plain_calculated,
                                 &output_length_tmp);
    assert(FSP_SUCCESS == err);
    p_output = &(plain_calculated[output_length_tmp]);
 /* Output remaining text and verify the tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, p_output, &output_length_tmp, mac,
CCM_MAC_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### AES-CMAC Example

This is an example of AES-CMAC signature/verification.

```
#define CMAC_MESSAGE_LEN (10)
#define CMAC_MAC_LEN (16)
void r_rsip_example_cmac ()
{
 fsp_err_t err;
 static const uint8_t message[CMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0xf0, 0x8f, 0x89, 0x08, 0x75, 0xe1, 0x39, 0x48, 0x04, 0x89
    };
    uint8_t mac[CMAC_MAC_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_AES_256
)] BSP_ALIGN_VARIABLE(4);
```

```
rsip_wrapped_key_t wrapped_key =
   {
      .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_key_value
   };
/* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* Generate a aes 256-bit key */
   err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
   assert(FSP_SUCCESS == err);
/* CMAC initialize for encrypt */
   err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
   assert(FSP_SUCCESS == err);
/* Input message */
   err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
   assert(FSP_SUCCESS == err);
/* Output remaining text and generate a tag */
   err = R_RSIP_AES_MAC_SignFinish(&g_rsip_ctrl, mac);
   assert(FSP_SUCCESS == err);
/* CMAC initialize for decrypt */
   err = R_RSIP_AES_MAC_Init(&g_rsip_ctrl, RSIP_AES_MAC_MODE_CMAC, &wrapped_key);
   assert(FSP_SUCCESS == err);
/* Input message */
   err = R_RSIP_AES_MAC_Update(&g_rsip_ctrl, message, CMAC_MESSAGE_LEN);
   assert(FSP_SUCCESS == err);
/* Verify the tag */
   err = R_RSIP_AES_MAC_VerifyFinish(&g_rsip_ctrl, mac, CMAC_MAC_LEN);
   assert(FSP_SUCCESS == err);
/* Close the driver */
   err = R_RSIP_Close(&g_rsip_ctrl);
   assert(FSP_SUCCESS == err);
}
```

### ChaCha20 Example

This is an example of ChaCha20 encryption/decryption.

```
#define CHACHA20_BLOCK_LEN (64)
#define CHACHA20_TEXT_LEN (114)
#define CHACHA20_NONCE_LEN (12)
void r_rsip_example_chacha ()
{
 fsp_err_t            err;
 static const uint8_t plain[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x4c, 0x61, 0x64, 0x69, 0x65, 0x73, 0x20, 0x61, 0x6e, 0x64, 0x20, 0x47, 0x65,
0x6e, 0x74, 0x6c,
       0x65, 0x6d, 0x65, 0x6e, 0x20, 0x6f, 0x66, 0x20, 0x74, 0x68, 0x65, 0x20, 0x63,
0x6c, 0x61, 0x73,
       0x73, 0x20, 0x6f, 0x66, 0x20, 0x27, 0x39, 0x39, 0x3a, 0x20, 0x49, 0x66, 0x20,
0x49, 0x20, 0x63,
       0x6f, 0x75, 0x6c, 0x64, 0x20, 0x6f, 0x66, 0x66, 0x65, 0x72, 0x20, 0x79, 0x6f,
0x75, 0x20, 0x6f,
       0x6e, 0x6c, 0x79, 0x20, 0x6f, 0x6e, 0x65, 0x20, 0x74, 0x69, 0x70, 0x20, 0x66,
0x6f, 0x72, 0x20,
       0x74, 0x68, 0x65, 0x20, 0x66, 0x75, 0x74, 0x75, 0x72, 0x65, 0x2c, 0x20, 0x73,
0x75, 0x6e, 0x73,
       0x63, 0x72, 0x65, 0x65, 0x6e, 0x20, 0x77, 0x6f, 0x75, 0x6c, 0x64, 0x20, 0x62,
0x65, 0x20, 0x69,
       0x74, 0x2e,
    };
 static const uint8_t nonce[CHACHA20_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
    };
    uint32_t counter = 0;
    uint8_t cipher_calculated[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[CHACHA20_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t input_length      = 0;
    uint32_t output_length     = 0;
```

```
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_CHACHA20)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_CHACHA20, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a ChaCha20 key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
/* Encrypt plaintext with one-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Init(&g_rsip_ctrl, &wrapped_key, nonce, counter);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) and output encrypted text */
    err = R_RSIP_ChaCha20_Update(&g_rsip_ctrl, plain, CHACHA20_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining encrypted text */
    err = R_RSIP_ChaCha20_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp);
    assert(FSP_SUCCESS == err);
/* Decrypt ciphertext with multi-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Init(&g_rsip_ctrl, &wrapped_key, nonce, counter);
    assert(FSP_SUCCESS == err);
    input_length = 0;
```

```
    output_length = 0;
 /* Input ciphertext (first 64 bytes) and output decrypted text */
    input_length_tmp = CHACHA20_BLOCK_LEN;
    err             = R_RSIP_ChaCha20_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    assert(FSP_SUCCESS == err);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
 /* Input remaining ciphertext and output decrypted text */
    input_length_tmp = CHACHA20_TEXT_LEN - input_length;
    err             = R_RSIP_ChaCha20_Update(&g_rsip_ctrl,
                                             cipher_calculated + input_length,
                                             input_length_tmp,
                                             plain_calculated + output_length,
                                             &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining decrypted text */
    err = R_RSIP_ChaCha20_Finish(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, CHACHA20_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ChaCha20-Poly1305 Example

This is an example of ChaCha20-Poly1305 encryption/decryption.

```
#define CHACHA_POLY_BLOCK_LEN (64)
#define CHACHA_POLY_TEXT_LEN (114)
#define CHACHA_POLY_NONCE_LEN (12)
#define CHACHA_POLY_AAD_LEN (12)
#define CHACHA_POLY_TAG_LEN (16)
void r_rsip_example_chacha_poly ()
{
 fsp_err_t            err;
 static const uint8_t plain[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x4c, 0x61, 0x64, 0x69, 0x65, 0x73, 0x20, 0x61, 0x6e, 0x64, 0x20, 0x47, 0x65,
0x6e, 0x74, 0x6c,
       0x65, 0x6d, 0x65, 0x6e, 0x20, 0x6f, 0x66, 0x20, 0x74, 0x68, 0x65, 0x20, 0x63,
0x6c, 0x61, 0x73,
       0x73, 0x20, 0x6f, 0x66, 0x20, 0x27, 0x39, 0x39, 0x3a, 0x20, 0x49, 0x66, 0x20,
0x49, 0x20, 0x63,
       0x6f, 0x75, 0x6c, 0x64, 0x20, 0x6f, 0x66, 0x66, 0x65, 0x72, 0x20, 0x79, 0x6f,
0x75, 0x20, 0x6f,
       0x6e, 0x6c, 0x79, 0x20, 0x6f, 0x6e, 0x65, 0x20, 0x74, 0x69, 0x70, 0x20, 0x66,
0x6f, 0x72, 0x20,
       0x74, 0x68, 0x65, 0x20, 0x66, 0x75, 0x74, 0x75, 0x72, 0x65, 0x2c, 0x20, 0x73,
0x75, 0x6e, 0x73,
       0x63, 0x72, 0x65, 0x65, 0x6e, 0x20, 0x77, 0x6f, 0x75, 0x6c, 0x64, 0x20, 0x62,
0x65, 0x20, 0x69,
       0x74, 0x2e,
    };
 static const uint8_t nonce[CHACHA_POLY_NONCE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
    };
 static const uint8_t aad[CHACHA_POLY_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x50, 0x51, 0x52, 0x53, 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
    };
```

```
    uint8_t cipher_calculated[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[CHACHA_POLY_TEXT_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t tag[CHACHA_POLY_TAG_LEN] BSP_ALIGN_VARIABLE(4)                = {0};
    uint32_t input_length      = 0;
    uint32_t output_length    = 0;
    uint32_t input_length_tmp = 0;
    uint32_t output_length_tmp = 0;
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_CHACHA20)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
    {
       .type = RSIP_KEY_TYPE_CHACHA20, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a ChaCha20 key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
/* Encrypt plaintext with one-shot update */
 /* Initialize ChaCha20-Poly1305 operation */
    err = R_RSIP_ChaCha20_Poly1305_Init(&g_rsip_ctrl,
 RSIP_CHACHA20_POLY1305_MODE_ENC,
                                     &wrapped_key,
                                     nonce,
                                     CHACHA_POLY_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_ChaCha20_Poly1305_AADUpdate(&g_rsip_ctrl, aad, CHACHA_POLY_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext (one-shot) and output encrypted text */
    err = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
```

```
                                        plain,
                                        CHACHA_POLY_TEXT_LEN,
                                    cipher_calculated,
                                    &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining encrypted text and generate a tag */
    err = R_RSIP_ChaCha20_Poly1305_Finish(&g_rsip_ctrl, cipher_calculated +
output_length, &output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
/* Decrypt ciphertext with multi-shot update */
 /* Initialize ChaCha20 operation */
    err = R_RSIP_ChaCha20_Poly1305_Init(&g_rsip_ctrl,
 RSIP_CHACHA20_POLY1305_MODE_DEC,
                                        &wrapped_key,
                                        nonce,
                                        CHACHA_POLY_NONCE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_ChaCha20_Poly1305_AADUpdate(&g_rsip_ctrl, aad, CHACHA_POLY_AAD_LEN);
    assert(FSP_SUCCESS == err);
    input_length = 0;
    output_length = 0;
 /* Input ciphertext (first 64 bytes) and output decrypted text */
    input_length_tmp = CHACHA_POLY_BLOCK_LEN;
    err             = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
                                                      cipher_calculated + input_length,
                                                      input_length_tmp,
                                                      plain_calculated + output_length,
                                                      &output_length_tmp);
    assert(FSP_SUCCESS == err);
    input_length += input_length_tmp;
    output_length += output_length_tmp;
 /* Input remaining ciphertext and output decrypted text */
```

```
    input_length_tmp = CHACHA_POLY_TEXT_LEN - input_length;
    err              = R_RSIP_ChaCha20_Poly1305_Update(&g_rsip_ctrl,
                                                       cipher_calculated + input_length,
                                                       input_length_tmp,
                                                       plain_calculated + output_length,
                                                       &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining decrypted text and verify the tag*/
    err = R_RSIP_ChaCha20_Poly1305_Verify(&g_rsip_ctrl,
                                          plain_calculated + output_length,
                                          &output_length_tmp,
                                          tag,
                                          CHACHA_POLY_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare plain and plain_calculated */
    assert(0 == memcmp(plain, plain_calculated, CHACHA_POLY_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ECC Example

This is an example of ECDSA signature generation/verification.

```
#define SECP256R1_KEY_SIZE (32)
void r_rsip_example_ecc ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
    uint8_t hash[SECP256R1_KEY_SIZE] BSP_ALIGN_VARIABLE(4)          = {0};
    uint8_t signature[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
```

```
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_ECDSA_Verify(&g_rsip_ctrl, &wrapped_public_key, hash, signature);
    assert(FSP_SUCCESS == err);
```

```
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### RSA Example

This is an example of RSA encryption/decryption (RSAES-OAEP) and RSA signature generation/verification (RSASSA-PSS).

```
#define SHA256_HASH_SIZE (32)
#define RSA_2048_KEY_SIZE (256)
void r_rsip_example_rsa ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE(4) = "sample";
 static const uint8_t label[] BSP_ALIGN_VARIABLE(4) = "label";
    uint8_t cipher_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t plain_calculated_length = 0;
    uint8_t hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)       = {0};
    uint8_t signature[RSA_2048_KEY_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_RSA_2048_PUBLIC, .p_value = &wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(RSIP_KEY_TYPE_RSA_2048_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
```

```
      .type = RSIP_KEY_TYPE_RSA_2048_PRIVATE, .p_value = &wrapped_private_key_value
   };
/* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* Generate a key pair of RSA-2048 */
   err =
R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key, &wrapped_private_key);
   assert(FSP_SUCCESS == err);
/*
 * RSAES-OAEP
 */
/* Encrypt a plaintext */
   err = R_RSIP_RSAES_OAEP_Encrypt(&g_rsip_ctrl,
                                   &wrapped_public_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                   message,
sizeof(message),
                                   cipher_calculated);
   assert(FSP_SUCCESS == err);
/* Decrypt a ciphertext using the same key */
   err = R_RSIP_RSAES_OAEP_Decrypt(&g_rsip_ctrl,
                                   &wrapped_private_key,
RSIP_HASH_TYPE_SHA256,
RSIP_MGF_TYPE_MGF1_SHA256,
                                   label,
sizeof(label),
                                  cipher_calculated,
                                  plain_calculated,
                                   &plain_calculated_length,
sizeof(plain_calculated));
```

```
    assert(FSP_SUCCESS == err);
 /* Verify calculated plaintext */
    assert(sizeof(message) == plain_calculated_length);
    assert(0 == memcmp(message, plain_calculated, plain_calculated_length));
 /*
  * RSASSA-PSS
  */
 /* Generate the message hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature */
    err = R_RSIP_RSASSA_PSS_Sign(&g_rsip_ctrl,
                                 &wrapped_private_key,
 RSIP_HASH_TYPE_SHA256,
 RSIP_MGF_TYPE_MGF1_SHA256,
 RSIP_RSA_SALT_LENGTH_AUTO,
                                 hash,
                                 signature);
    assert(FSP_SUCCESS == err);
 /* Verify the signature*/
    err = R_RSIP_RSASSA_PSS_Verify(&g_rsip_ctrl,
                                  &wrapped_public_key,
 RSIP_HASH_TYPE_SHA256,
 RSIP_MGF_TYPE_MGF1_SHA256,
 RSIP_RSA_SALT_LENGTH_AUTO,
                                   hash,
                                   signature);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HASH Example

This is an example of calculating the SHA-256 message digest.

```
#define SHA256_HASH_SIZE (32)
void r_rsip_example_hash ()
{
 fsp_err_t err;
 static const uint8_t message[] BSP_ALIGN_VARIABLE (4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
       0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    };
 static const uint8_t hash[] BSP_ALIGN_VARIABLE (4) =
    {
       0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
       0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
    };
    uint8_t digest_single[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t digest_multi[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4) = {0};
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate SHA-256 digests with single-part API */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, message,
sizeof(message), digest_single);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_single, hash, SHA256_HASH_SIZE));
 /* (2) Calculate SHA-256 digests with multi-part API */
    err = R_RSIP_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
```

```
    err = R_RSIP_SHA_Update(&g_rsip_ctrl, message, sizeof(message));
    assert(FSP_SUCCESS == err);
    err = R_RSIP_SHA_Finish(&g_rsip_ctrl, digest_multi);
    assert(FSP_SUCCESS == err);
 /* Compare digest and hash */
    assert(0 == memcmp(digest_multi, hash, SHA256_HASH_SIZE));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### HMAC Example

This is an example of HMAC signature/verification.

```
#define HMAC_MESSAGE_LEN (34)
#define HMAC_HASH_LEN (32)
void r_rsip_example_hmac ()
{
 fsp_err_t err;
 static const uint8_t message[HMAC_MESSAGE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x53, 0x61, 0x6D, 0x70, 0x6C, 0x65, 0x20, 0x6D,
       0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x20, 0x66,
       0x6F, 0x72, 0x20, 0x6B, 0x65, 0x79, 0x6C, 0x65,
       0x6E, 0x3C, 0x62, 0x6C, 0x6F, 0x63, 0x6B, 0x6C,
       0x65, 0x6E
    };
    uint8_t hash_1[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t hash_2[HMAC_HASH_LEN] BSP_ALIGN_VARIABLE(4) = {0};
 /* Prepare wrapped key data area */

uint8_t            wrapped_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_key =
```

```
    {
        .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_key_value
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a HMAC-SHA 256-bit key */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* (1) Calculate HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to finish */
    err = R_RSIP_HMAC_Compute(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1);
    assert(FSP_SUCCESS == err);
 /* (2) Verify HMAC-SHA-256 hash with single-part API */
 /* HMAC initialize to verify */
    err = R_RSIP_HMAC_Verify(&g_rsip_ctrl, &wrapped_key, message, HMAC_MESSAGE_LEN,
hash_1, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* (3) Calculate HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC generation */
    err = R_RSIP_HMAC_SignFinish(&g_rsip_ctrl, hash_2);
    assert(FSP_SUCCESS == err);
 /* (4) Verify HMAC-SHA-256 hash with multi-part API */
 /* HMAC initialize */
    err = R_RSIP_HMAC_Init(&g_rsip_ctrl, &wrapped_key);
    assert(FSP_SUCCESS == err);
 /* Inputs message */
```

```
    err = R_RSIP_HMAC_Update(&g_rsip_ctrl, message, HMAC_MESSAGE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalizes a HMAC verification */
    err = R_RSIP_HMAC_VerifyFinish(&g_rsip_ctrl, hash_2, HMAC_HASH_LEN);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
 /* Compare hash_1 and hash_2 */
    assert(0 == memcmp(hash_1, hash_2, HMAC_HASH_LEN));
}
```

### PKI Example

This is an example that verifies the public key certificate, imports the peer's public key, and calculates the shared secret.

```
#define SHA256_HASH_SIZE (32)
#define SECP256R1_KEY_SIZE (32)
/* Simplified public key certificate structure */
typedef struct st_cert
{
    uint8_t public_key_param_qx[SECP256R1_KEY_SIZE];
    uint8_t public_key_param_qy[SECP256R1_KEY_SIZE];
} cert_t;
void r_rsip_example_pki ()
{
 fsp_err_t err;
 /* Prepare data area for generating certificate and shared secret */
    cert_t               cert;
    uint8_t              raw_public_key[SECP256R1_KEY_SIZE * 2]
BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t              hash[SHA256_HASH_SIZE] BSP_ALIGN_VARIABLE(4)
= {0};
    uint8_t              signature[SECP256R1_KEY_SIZE * 2]
```

```
BSP_ALIGN_VARIABLE(4)      = {0};
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare wrapped key data area for CA (Virtually prepared for local validation) */
    uint8_t
wrapped_ca_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_ca_public_key_value
    };
    uint8_t
wrapped_ca_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_ca_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_ca_private_key_value
    };
 /* Prepare wrapped key data area for peer (Virtually prepared for local validation)
*/
    uint8_t
wrapped_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_peer_public_key_value
    };
    uint8_t
wrapped_peer_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
```

```
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_peer_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_peer_private_key_value
    };
 /* Prepare wrapped key data area for Local device */
    uint8_t
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 /* Prepare wrapped key data area for verified peer public key */
    uint8_t wrapped_verified_peer_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
 RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_verified_peer_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_verified_peer_public_key_value
    };
```

```
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate key pairs for CA, peer, and Local device */
 /* Generate a key pair of secp256r1 for CA (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_ca_public_key,
&wrapped_ca_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for peer (Virtually generated for local
validation) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_peer_public_key,
&wrapped_peer_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device (Assume product with RSIP
driver) */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
/* CA process :
 * In the following, generate certificate and signature of peer public key */
 /* Export raw peer public key (Qx|Qy) and store it in cert */
    err = R_RSIP_PublicKeyExport(&wrapped_peer_public_key, raw_public_key);
    assert(FSP_SUCCESS == err);
    memcpy(cert.public_key_param_qx, &raw_public_key[0], SECP256R1_KEY_SIZE);
    memcpy(cert.public_key_param_qy, &raw_public_key[SECP256R1_KEY_SIZE],
SECP256R1_KEY_SIZE);
 /* Generate the hash of cert and store it in hash */
    err = R_RSIP_SHA_Compute(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256, (uint8_t *) &cert,
sizeof(cert), hash);
    assert(FSP_SUCCESS == err);
 /* Generate the signature of hash with CA private key and store it in signature */
    err = R_RSIP_ECDSA_Sign(&g_rsip_ctrl, &wrapped_ca_private_key, hash, signature);
    assert(FSP_SUCCESS == err);
```

```
/* Local device process :
 * In the following, verify the CA certificate, import peer public key, and generate
shared secret */
 /* Verify the signature of cert with CA public key */
    err = R_RSIP_PKI_ECDSA_CertVerify(&g_rsip_ctrl, &wrapped_ca_public_key, hash,
signature);
    assert(FSP_SUCCESS == err);
 /* Import verified peer public key with wrapped format */
    err = R_RSIP_PKI_CertKeyImport(&g_rsip_ctrl,
                                   (uint8_t *) &cert,
 sizeof(cert),
                                   cert.public_key_param_qx,
                                   SECP256R1_KEY_SIZE,
                                   cert.public_key_param_qy,
                                   SECP256R1_KEY_SIZE,
 RSIP_HASH_TYPE_SHA256,
                                   &wrapped_verified_peer_public_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the Local private key and the verified peer public
key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl,
                               &wrapped_local_private_key,
                               &wrapped_verified_peer_public_key,
                               &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ECDH (with plain public key) Example

This is an example of calculating the shared secret using a plaintext key that has been received from somewhere else.

```
```

```
#define SECP256R1_KEY_SIZE (32)
#define COUNTER_BYTE_LEN (4)
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (64)
#define WRAPPED_IV_BYTE_SIZE (36)
void r_rsip_example_ecdh ()
{
 fsp_err_t err;
 /* Plain public key data for ECC secp256r1 */
 static const uint8_t plain_public_key[SECP256R1_KEY_SIZE * 2] BSP_ALIGN_VARIABLE(4)
=
    {
       0x60, 0xfe, 0xd4, 0xba, 0x25, 0x5a, 0x9d, 0x31, 0xc9, 0x61, 0xeb, 0x74, 0xc6,
0x35, 0x6d, 0x68,
       0xc0, 0x49, 0xb8, 0x92, 0x3b, 0x61, 0xfa, 0x6c, 0xe6, 0x69, 0x62, 0x2e, 0x60,
0xf2, 0x9f, 0xb6,
       0x79, 0x03, 0xfe, 0x10, 0x08, 0xb8, 0xbc, 0x99, 0xa4, 0x1a, 0xe9, 0xe9, 0x56,
0x28, 0xbc, 0x64,
       0xf2, 0xf1, 0xb2, 0x0c, 0x2d, 0x7e, 0x9f, 0x51, 0x77, 0xa3, 0xc2, 0x94, 0xd4,
0x46, 0x22, 0x99,
    };
 /* Prepare data area for local key pair and shared secret */
    uint8_t
wrapped_local_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_local_public_key_value
    };
    uint8_t
wrapped_local_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
```

```
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_local_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_local_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm[KDF_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm = (rsip_wrapped_dkm_t *) wrapped_dkm;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-SHA operation */
    uint32_t            counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
 /* Generate a key pair of secp256r1 for Local device */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_local_public_key,
&wrapped_local_private_key);
    assert(FSP_SUCCESS == err);
```

```
 /* Generate shared secret from the Local private key and the plain public key */
    err = R_RSIP_ECDH_PlainKeyAgree(&g_rsip_ctrl, &wrapped_local_private_key,
plain_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* In the following, Generate DKM (Derived Keying Material) by calculating KDF-
SHA[counter|Z|FixedInfo] as an example, where Z is shared secret obtained above */
    counter = 0x00000000;
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES from the DKM */
 /* Derive the AES-256 wrapped key from the 1st byte position of the DKM */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm, 0, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 1st byte position of the DKM */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   0,
```

```
                                    NULL,
                                    wrapped_iv);
    assert(FSP_SUCCESS == err);
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### KDF-SHA Example

This is an example of One-Step Key Derivation (NIST SP 800-56C) using KDF-SHA.

```
#define COUNTER_BYTE_LEN (4)
#define FIXED_INFO_BYTE_LEN (16)
#define KDF_SHA256_DKM_BYTE_SIZE (64)
#define CONCAT_DKM_BYTE_SIZE KDF_SHA256_DKM_BYTE_SIZE * 2
#define WRAPPED_IV_BYTE_SIZE (36)
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (16)
void r_rsip_example_kdf_sha ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
```

```
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped DKM (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_DKM_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_SHA256_DKM_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the DKM */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-SHA operation */
    uint32_t             counter = 0;
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
```

```
      0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
      0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
   };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
   {
      0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
   };
   uint8_t tag[GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
   uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
   uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
   uint32_t output_length    = 0;
   uint32_t output_length_tmp = 0;
 /* Initialize the driver */
   err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
   assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
 /* Generate a key pair of secp256r1 to generate the shared secret */
   err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
   assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
   err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
   assert(FSP_SUCCESS == err);
/* One-Step Key Derivation (NIST SP 800-56C)
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
   counter = 0x00000000;
   counter++;
```

```
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM1 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-SHA[counter|Z|FixedInfo], where Z is shared
secret */
    counter++;
 /* Initialize KDF-SHA operation */
    err = R_RSIP_KDF_SHA_Init(&g_rsip_ctrl, RSIP_HASH_TYPE_SHA256);
    assert(FSP_SUCCESS == err);
 /* Input counter as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, (uint8_t *) &counter,
COUNTER_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_SHA_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Input Fixed Info as a plaintext */
    err = R_RSIP_KDF_SHA_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-SHA operation and generate DKM2 (Derived Keying Material) */
    err = R_RSIP_KDF_SHA_Finish(&g_rsip_ctrl, p_wrapped_dkm2);
```

```
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_DKM_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In One-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   16,
                                   NULL,
                                   wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
```

```
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, cipher_calculated, GCM_TEXT_LEN,
plain_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### KDF-HMAC Example

This is an example of Two-Step Key Derivation (NIST SP 800-56C) using KDF-HMAC.

```
#define INITIAL_VALUE_BYTE_LEN (32)
#define FIXED_INFO_BYTE_LEN (16)
#define HMAC_SHA256_KEY_LEN (32)
#define KDF_HMAC_SHA256_MAC_BYTE_SIZE (64)
#define CONCAT_MAC_BYTE_SIZE KDF_HMAC_SHA256_MAC_BYTE_SIZE * 2
#define WRAPPED_IV_BYTE_SIZE (36)
#define GCM_BLOCK_LEN (16)
#define GCM_TEXT_LEN (31)
#define GCM_AAD_LEN (8)
#define GCM_TAG_LEN (16)
void r_rsip_example_kdf_hmac ()
{
 fsp_err_t err;
 /* Prepare data area for key pair and shared secret */
    uint8_t
wrapped_public_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_public_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value =
&wrapped_public_key_value
    };
    uint8_t
wrapped_private_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_private_key =
    {
       .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value =
&wrapped_private_key_value
    };
```

```
 rsip_wrapped_secret_t wrapped_secret;
 /* Prepare data area for wrapped key as a salt */

uint8_t            wrapped_salt_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_salt =
    {
       .type = RSIP_KEY_TYPE_HMAC_SHA256, .p_value = &wrapped_salt_value
    };
 /* Prepare data area for wrapped KDK (Key Derivation Key) */

uint8_t            wrapped_kdk_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_KEY_TYPE_HMAC_SHA256)] BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_kdk =
    {
       .type = RSIP_KEY_TYPE_KDF_HMAC_SHA256, .p_value = &wrapped_kdk_value
    };
 /* Prepare data area for wrapped MAC (Derived Keying Material) */
    uint8_t             wrapped_dkm1[CONCAT_MAC_BYTE_SIZE];
    uint8_t             wrapped_dkm2[KDF_HMAC_SHA256_MAC_BYTE_SIZE];
 rsip_wrapped_dkm_t * p_wrapped_dkm1 = (rsip_wrapped_dkm_t *) wrapped_dkm1;
 rsip_wrapped_dkm_t * p_wrapped_dkm2 = (rsip_wrapped_dkm_t *) wrapped_dkm2;
 /* Prepare wrapped key and wrapped IV data area that will be derived from the MAC */
    uint8_t
wrapped_derived_key_value[RSIP_BYTE_SIZE_WRAPPED_KEY(
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256)]
    BSP_ALIGN_VARIABLE(4);
 rsip_wrapped_key_t wrapped_derived_key =
    {
       .type = RSIP_KEY_TYPE_AES_256, .p_value = &wrapped_derived_key_value
    };
    uint8_t wrapped_iv[WRAPPED_IV_BYTE_SIZE];
 /* Input data for KDF-HMAC operation */
 static const uint8_t initial_value[INITIAL_VALUE_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
```

```
    {
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
       0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa, 0x55, 0xaa,
    };
 static const uint8_t fixed_info[FIXED_INFO_BYTE_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    };
 /* Prepare a data area for AES-GCM operation (For validation check) */
 static const uint8_t plain[GCM_TEXT_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
       0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e,
    };
 static const uint8_t aad[GCM_AAD_LEN] BSP_ALIGN_VARIABLE(4) =
    {
       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    };
    uint8_t tag[GCM_TAG_LEN] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t cipher_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint8_t plain_calculated[GCM_BLOCK_LEN * 2] BSP_ALIGN_VARIABLE(4) = {0};
    uint32_t output_length    = 0;
    uint32_t output_length_tmp = 0;
 /* Initialize the driver */
    err = R_RSIP_Open(&g_rsip_ctrl, &g_rsip_cfg);
    assert(FSP_SUCCESS == err);
/* In the following, generate shared secret from the locally generated key pair
 * For more accurate process of generating the shared secret, refer to the PKI
example code */
```

```
 /* Generate a key pair of secp256r1 to generate the shared secret */
    err = R_RSIP_KeyPairGenerate(&g_rsip_ctrl, &wrapped_public_key,
&wrapped_private_key);
    assert(FSP_SUCCESS == err);
 /* Generate shared secret from the private key and public key */
    err = R_RSIP_ECDH_KeyAgree(&g_rsip_ctrl, &wrapped_private_key,
&wrapped_public_key, &wrapped_secret);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 1 : Randomness Extraction
 * Generate KDK (Key Derivation Key) */
/* In the following, calculate KDK = KDF-HMAC[salt, Z], where Z is shared secret */
 /* Generate a HMAC-SHA 256 key as a salt */
    err = R_RSIP_KeyGenerate(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Initialize KDF-HMAC operation (Key setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_salt);
    assert(FSP_SUCCESS == err);
 /* Input shared secret as a ciphertext */
    err = R_RSIP_KDF_HMAC_ECDHSecretUpdate(&g_rsip_ctrl, &wrapped_secret);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Convert the wrapped MAC to wrapped HMAC-SHA256 key, which is called KDK (Key
Derivation Key) */
    err = R_RSIP_KDF_HMAC_DKMKeyImport(&g_rsip_ctrl, p_wrapped_dkm1,
HMAC_SHA256_KEY_LEN, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
/* Two-Step Key Derivation (NIST SP 800-56C) Step 2 : Key Expansion
 * Generate DKM (Derived Keying Material) */
/* In the following, calculate K(1) = KDF-HMAC[KDK, K(0)|FixedInfo], where K(0) = IV
*/
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
```

```
    assert(FSP_SUCCESS == err);
 /* Input K(0) = IV as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, initial_value,
INITIAL_VALUE_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
/* In the following, calculate K(2) = KDF-HMAC[KDK, K(1)|FixedInfo] */
 /* Initialize KDF-HMAC operation (KDK setting) */
    err = R_RSIP_KDF_HMAC_Init(&g_rsip_ctrl, &wrapped_kdk);
    assert(FSP_SUCCESS == err);
 /* Input K(1) as a ciphertext */
    err = R_RSIP_KDF_HMAC_DKMUpdate(&g_rsip_ctrl, p_wrapped_dkm1);
    assert(FSP_SUCCESS == err);
 /* Input FixedInfo as a plaintext */
    err = R_RSIP_KDF_HMAC_Update(&g_rsip_ctrl, fixed_info, FIXED_INFO_BYTE_LEN);
    assert(FSP_SUCCESS == err);
 /* Finalize KDF-HMAC operation and generate wrapped MAC */
    err = R_RSIP_KDF_HMAC_SignFinish(&g_rsip_ctrl, p_wrapped_dkm2);
    assert(FSP_SUCCESS == err);
/* In the following, concatenate K(1) and K(2) to obtain DKM */
 /* Concatenate K(2) to the end of K(1), and obtain DKM as p_wrapped_dkm1 */
    err = R_RSIP_KDF_DKMConcatenate(p_wrapped_dkm1, p_wrapped_dkm2,
CONCAT_MAC_BYTE_SIZE);
    assert(FSP_SUCCESS == err);
/* In the following, derive wrapped key and wrapped IV for AES operation from the DKM
 * In Two-Step Key Derivation, it is used from the first position of the concatenated
DKM
 * Here, it is used from the 16th position as an example of how to use this API */
 /* Derive the AES-256 wrapped key from the 16th byte position of the concatenated
```

```
DKM1 */
    err =
 R_RSIP_KDF_DerivedKeyImport(&g_rsip_ctrl, p_wrapped_dkm1, 16, &wrapped_derived_key);
    assert(FSP_SUCCESS == err);
 /* Derive the wrapped IV for AES from the 16th byte position of the concatenated
DKM1 */
    err = R_RSIP_KDF_DerivedIVWrap(&g_rsip_ctrl,
                                   p_wrapped_dkm1,
 RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE,
                                   16,
                                   NULL,
                                   wrapped_iv);
    assert(FSP_SUCCESS == err);
/* In the following, verify the validity of wrapped_derived_key and wrapped_iv using
AES-GCM operation */
 /* Initialize AES-GCM operation using wrapped_derived_key and wrapped_iv */
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input plaintext and output ciphertext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, plain, GCM_TEXT_LEN,
cipher_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining ciphertext and generate a tag */
    err = R_RSIP_AES_AEAD_Finish(&g_rsip_ctrl, cipher_calculated + output_length,
&output_length_tmp, tag);
    assert(FSP_SUCCESS == err);
    output_length = 0;
 /* Initialize AES-GCM operation using derived wrapped key and wrapped IV above */
```

```
    err =
 R_RSIP_AES_AEAD_Init(&g_rsip_ctrl, RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV,
&wrapped_derived_key, wrapped_iv, 0);
    assert(FSP_SUCCESS == err);
 /* Input AAD */
    err = R_RSIP_AES_AEAD_AADUpdate(&g_rsip_ctrl, aad, GCM_AAD_LEN);
    assert(FSP_SUCCESS == err);
 /* Input ciphertext and output plaintext */
    err = R_RSIP_AES_AEAD_Update(&g_rsip_ctrl, cipher_calculated, GCM_TEXT_LEN,
plain_calculated, &output_length_tmp);
    assert(FSP_SUCCESS == err);
    output_length += output_length_tmp;
 /* Output remaining plaintext and verify a tag */
    err = R_RSIP_AES_AEAD_Verify(&g_rsip_ctrl, plain_calculated + output_length,
&output_length_tmp, tag, GCM_TAG_LEN);
    assert(FSP_SUCCESS == err);
 /* Compare the plaintext data before encryption with the decrypted data */
    assert(0 == memcmp(plain, plain_calculated, GCM_TEXT_LEN));
 /* Close the driver */
    err = R_RSIP_Close(&g_rsip_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | rsip_sha_handle_t |
| struct | rsip_hmac_handle_t |
| struct | rsip_kdf_sha_handle_t |
| struct | rsip_kdf_hmac_handle_t |
| struct | rsip_verified_cert_info_t |
| struct | rsip_wrapped_secret_t |
| struct | rsip_instance_ctrl_t |

## Macros

| | |
|---|---|
| #define | RSIP_BYTE_SIZE_PLAIN_KEY(key_type) |
| #define | RSIP_BYTE_SIZE_ENCRYPTED_KEY(key_type) |
| #define | RSIP_BYTE_SIZE_WRAPPED_KEY(key_type) |
| #define | FSP_ERR_SB_INTERNAL_FAIL<br>An internal failure. |
| #define | FSP_ERR_SB_INVALID_ARG<br>An invalid argument was entered. |
| #define | FSP_ERR_SB_UNSUPPORTED_FUNCTION<br>Unsupported function executed. |
| #define | FSP_ERR_SB_INVALID_ALIGNMENT<br>Data entered with incorrect alignment. |
| #define | FSP_ERR_SB_SAME_IMAGE_VERSION<br>Same image version. |
| #define | FSP_ERR_SB_LOWER_IMAGE_VERSION<br>Lower image version. |
| #define | FSP_ERR_SB_MANI_INVALID_MAGIC<br>An invalid magic number is set. |
| #define | FSP_ERR_SB_MANI_UNSUPPORTED_VERSION<br>Unsupported version is set. |
| #define | FSP_ERR_SB_MANI_OUT_OF_RANGE_LEN<br>Out of range TLV Length is set. |
| #define | FSP_ERR_SB_MANI_TLV_FIELD_ERR<br>Missing required TLV field. |

| | |
|---|---|
| #define | FSP_ERR_SB_MANI_TLV_INVALID_LEN<br>The length exceeding the end of the manifest is specified in length of the TLV field. |
| #define | FSP_ERR_SB_MANI_INVALID_IMAGE_LEN<br>An invalid image length is set. |
| #define | FSP_ERR_SB_MANI_MISMATCH_SIGN_ALGORITHM<br>There is a wrong combination of signature algorithms. |
| #define | FSP_ERR_SB_MANI_UNSUPPORTED_ALGORITHM<br>An algorithm was specified that the manifest does not support. |
| #define | FSP_ERR_SB_CRYPTO_FAIL<br>Cryptographic processing failure. |
| #define | FSP_ERR_SB_CRYPTO_AUTH_FAIL<br>Verification failed. |
| #define | FSP_ERR_SB_CRYPTO_UNSUPPORTED_ALGORITHM<br>Unsupported algorithm. |
| #define | FSP_ERR_SB_CRYPTO_RESOURCE_CONFLICT<br>CryptoIP is in use. |
| #define | FSP_ERR_SB_CRYPTO_PARAM_ERR<br>Parameter error. |

## Enumerations

| | |
|---|---|
| enum | rsip_byte_size_wrapped_key_t |
| enum | rsip_byte_size_encrypted_key_t |
| enum | rsip_byte_size_wrapped_dkm_t |

## Data Structure Documentation

### ◆ rsip_sha_handle_t

| struct rsip_sha_handle_t |
| --- |
| Working area for SHA functions. DO NOT MODIFY. |

### ◆ rsip_hmac_handle_t

| struct rsip_hmac_handle_t |
| --- |
| Working area for HMAC functions. DO NOT MODIFY. |

### ◆ rsip_kdf_sha_handle_t

| struct rsip_kdf_sha_handle_t |
| --- |
| Working area for KDF SHA functions. DO NOT MODIFY. |

### ◆ rsip_kdf_hmac_handle_t

| struct rsip_kdf_hmac_handle_t |
| --- |
| Working area for KDF HMAC functions. DO NOT MODIFY. |

### ◆ rsip_verified_cert_info_t

| struct rsip_verified_cert_info_t |
| --- |
| Verified certificate information |

### ◆ rsip_wrapped_secret_t

| struct rsip_wrapped_secret_t |
| --- |
| Wrapped ECDH secret structure for ECDH and KDF APIs. |

### ◆ rsip_instance_ctrl_t

| struct rsip_instance_ctrl_t |
| --- |
| RSIP private control block. DO NOT MODIFY. Initialization occurs when R_RSIP_Open() is called. |

## Macro Definition Documentation

### ◆ RSIP_BYTE_SIZE_PLAIN_KEY

| #define RSIP_BYTE_SIZE_PLAIN_KEY ( key_type) |
| --- |
| Returns the plain key size formatted for RSIP driver of the key type. |

### ◆ RSIP_BYTE_SIZE_ENCRYPTED_KEY

| #define RSIP_BYTE_SIZE_ENCRYPTED_KEY ( key_type) |
| --- |
| Returns the encrypted key size of the key type. |

### ◆ RSIP_BYTE_SIZE_WRAPPED_KEY

| #define RSIP_BYTE_SIZE_WRAPPED_KEY ( key_type) |
| --- |
| Returns the wrapped key size of the key type. |

## Enumeration Type Documentation

### ◆ rsip_byte_size_wrapped_key_t

| enum rsip_byte_size_wrapped_key_t | |
|---|---|
| Byte size of wrapped key | |
| Enumerator | |
| RSIP_BYTE_SIZE_WRAPPED_KEY_AES_128 | AES-128. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_AES_192 | AES-192. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256 | AES-256. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_XTS_AES_128 | XTS-AES-128. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_XTS_AES_256 | XTS-AES-256. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_CHACHA20 | ChaCha20. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_PUBLIC | secp256r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP384R1_PUBLIC | secp384r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP521R1_PUBLIC | secp521r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PUBLIC | secp256k1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP256R1_PUBLIC | brainpoolP256r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP384R1_PUBLIC | brainpoolP384r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP512R1_PUBLIC | brainpoolP512r1 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_EDWARDS25519_PUBLIC | edwards25519 public key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_PRIVATE | secp256r1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP384R1_PRIVATE | secp384r1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP521R1_PRIVATE | secp521r1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PRIVATE | secp256k1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP256R1_PRIVATE | brainpoolP256r1 private key |

| | |
|---|---|
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP384R1_PRIVATE | brainpoolP384r1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOLP512R1_PRIVATE | brainpoolP512r1 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_EDWARDS25519_PRIVATE | edwards25519 private key |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_1024_PUBLIC | RSA-1024 public key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_2048_PUBLIC | RSA-2048 public key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_3072_PUBLIC | RSA-3072 public key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_4096_PUBLIC | RSA-4096 public key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_1024_PRIVATE | RSA-1024 private key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_2048_PRIVATE | RSA-2048 private key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_3072_PRIVATE | RSA-3072 private key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_RSA_4096_PRIVATE | RSA-4096 private key. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA1 | HMAC-SHA1. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA224 | HMAC-SHA224. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA256 | KDF HMAC-SHA256. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA384 | KDF HMAC-SHA384. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_KDF_HMAC_SHA512 | KDF HMAC-SHA512. |
| RSIP_BYTE_SIZE_WRAPPED_KEY_KUK | Key Update Key (KUK) |

### ◆ rsip_byte_size_encrypted_key_t

| enum rsip_byte_size_encrypted_key_t | |
|---|---|
| Byte size of encrypted key | |
| Enumerator | |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_128 | AES-128. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_192 | AES-192. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_AES_256 | AES-256. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_XTS_AES_128 | XTS-AES-128. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_XTS_AES_256 | XTS-AES-256. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_CHACHA20 | ChaCha20. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256R1_PUBLIC | secp256r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP384R1_PUBLIC | secp384r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP521R1_PUBLIC | secp521r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256K1_PUBLIC | secp256k1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP256R1_PUBLIC | brainpoolP256r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP384R1_PUBLIC | brainpoolP384r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP512R1_PUBLIC | brainpoolP512r1 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_EDWARDS25519_PUBLIC | edwards25519 public key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256R1_PRIVATE | secp256r1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP384R1_PRIVATE | secp384r1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP521R1_PRIVATE | secp521r1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_SECP256K1_PRIVATE | secp256k1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPO | brainpoolP256r1 private key |

| | |
|---|---|
| OLP256R1_PRIVATE | |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP384R1_PRIVATE | brainpoolP384r1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_BRAINPOOLP512R1_PRIVATE | brainpoolP512r1 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_ECC_EDWARDS25519_PRIVATE | edwards25519 private key |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_1024_PUBLIC | RSA-1024 public key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_2048_PUBLIC | RSA-2048 public key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_3072_PUBLIC | RSA-3072 public key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_4096_PUBLIC | RSA-4096 public key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_1024_PRIVATE | RSA-1024 private key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_2048_PRIVATE | RSA-2048 private key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_3072_PRIVATE | RSA-3072 private key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_RSA_4096_PRIVATE | RSA-4096 private key. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA1 | HMAC-SHA1. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA224 | HMAC-SHA224. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA256 | KDF HMAC-SHA256. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA384 | KDF HMAC-SHA384. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_KDF_HMAC_SHA512 | KDF HMAC-SHA512. |
| RSIP_BYTE_SIZE_ENCRYPTED_KEY_KUK | Key Update Key (KUK) |

### ◆ rsip_byte_size_wrapped_dkm_t

| enum rsip_byte_size_wrapped_dkm_t | |
|---|---|
| Byte size of wrapped key | |
| Enumerator | |
| RSIP_BYTE_SIZE_WRAPPED_DKM_HEADER | Header. |
| RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA256 | A block of SHA-256. |
| RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA384 | A block of SHA-384. |
| RSIP_BYTE_SIZE_WRAPPED_DKM_BLOCK_SHA512 | A block of SHA-384. |

## Function Documentation

### ◆ R_RSIP_Open()

fsp_err_t R_RSIP_Open ( rsip_ctrl_t *const *p_ctrl*, rsip_cfg_t const *const *p_cfg* )

Enables use of Renesas Secure IP functionality.

Implements rsip_api_t::open.

**State transition**

This API can only be executed in **STATE_INITIAL**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Internal key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Hardware initialization is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption or hardware fault is detected. |

***Note***

*This version does not have an optional feature to disable TRNG initialization.*

### ◆ R_RSIP_Close()

fsp_err_t R_RSIP_Close ( rsip_ctrl_t *const *p_ctrl*)

Disables use of Renesas Secure IP functionality.

Implements rsip_api_t::close.

**State transition**

This API can be executed in **except STATE_INITIAL**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_INITIAL |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_RandomNumberGenerate()

fsp_err_t R_RSIP_RandomNumberGenerate ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_random* )

Generates a 128-bit random number.

Implements rsip_api_t::randomNumberGenerate.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KeyGenerate()

fsp_err_t R_RSIP_KeyGenerate ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t *const *p_wrapped_key* )

Generates a wrapped symmetric key from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Implements rsip_api_t::keyGenerate.

**Conditions**

Argument key_type must be one of the following:

- RSIP_KEY_TYPE_AES_128, RSIP_KEY_TYPE_AES_256
- RSIP_KEY_TYPE_XTS_AES_128, RSIP_KEY_TYPE_XTS_AES_256
- RSIP_KEY_TYPE_HMAC_SHA224, RSIP_KEY_TYPE_HMAC_SHA256, RSIP_KEY_TYPE_HMAC_SHA384, RSIP_KEY_TYPE_HMAC_SHA512

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_KeyPairGenerate()

fsp_err_t R_RSIP_KeyPairGenerate ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t *const *p_wrapped_public_key*, rsip_wrapped_key_t *const *p_wrapped_private_key* )

Generates a wrapped asymmetric key pair from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Implements rsip_api_t::keyPairGenerate.

**Conditions**

Argument p_wrapped_public_key->type must be one of the following:

- RSIP_KEY_TYPE_ECC_*_PUBLIC
- RSIP_KEY_TYPE_RSA_*_PUBLIC Argument p_wrapped_private_key->type must be one of the following:
- RSIP_KEY_TYPE_ECC_*_PRIVATE
- RSIP_KEY_TYPE_RSA_*_PRIVATE Here, p_wrapped_public_key->type and p_wrapped_private_key->type must match. For example, if p_wrapped_public_key->type is RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, then p_wrapped_public_key->type must be RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_EncryptedKeyWrap()

fsp_err_t R_RSIP_EncryptedKeyWrap ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_key_update_key*, void const *const *p_initial_vector*, void const *const *p_encrypted_key*, rsip_wrapped_key_t *const *p_wrapped_key* )

Decrypts an encrypted user key with Key Update Key (KUK) and wrap it with the Hardware Unique Key (HUK).

Implements rsip_api_t::encryptedKeyWrap.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input KUK is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms

### ◆ R_RSIP_RFC3394_KeyWrap()

fsp_err_t R_RSIP_RFC3394_KeyWrap ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_kek*, rsip_wrapped_key_t const *const *p_wrapped_target_key*, uint8_t *const *p_rfc3394_wrapped_target_key* )

This function provides Key Wrap algorithm compliant with RFC3394. Using p_wrapped_kek to wrap p_wrapped_target_key, and output the result to p_rfc3394_wrapped_target_key.

Implements rsip_api_t::rfc3394_KeyWrap.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Conditions**

- Key type of p_wrapped_kek must be one of the following:
  - RSIP_KEY_TYPE_AES_128
  - RSIP_KEY_TYPE_AES_256
- Key type of p_wrapped_target_key must be one of the following:
  - RSIP_KEY_TYPE_AES_128
  - RSIP_KEY_TYPE_AES_256

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_ARGUMENT | Input key type or mode is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_RFC3394_KeyUnwrap()

fsp_err_t R_RSIP_RFC3394_KeyUnwrap ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_kek*, uint8_t const *const *p_rfc3394_wrapped_target_key*, rsip_wrapped_key_t *const *p_wrapped_target_key* )

This function provides Key Unwrap algorithm compliant with RFC3394. Using p_wrapped_kek to unwrap p_rfc3394_wrapped_target_key, and output the result to p_wrapped_target_key.

Implements rsip_api_t::rfc3394_KeyUnwrap.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Conditions**

- Key type of p_wrapped_kek must be one of the following:
  - RSIP_KEY_TYPE_AES_128
  - RSIP_KEY_TYPE_AES_256
- The third argument key_type represents the key type of p_rfc3394_wrapped_target_key must be one of the following:
  - RSIP_KEY_TYPE_AES_128
  - RSIP_KEY_TYPE_AES_256

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_ARGUMENT | Input key type or mode is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_PublicKeyExport()

fsp_err_t R_RSIP_PublicKeyExport ( rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t *const *p_raw_public_key* )

Exports public key parameters from a wrapped key.

Implements rsip_api_t::publicKeyExport.

Relative position of each elements in p_raw_public_key is shown in below:

- ECC (RSIP_KEY_TYPE_ECC_*) : Qx placed first and Qy placed after that.

| bit length | Qx | Qy |
|---|---|---|
| 256 | 0 | 32 |
| 384 | 0 | 48 |
| 512 | 0 | 64 |
| 521 | 0 | 66 |

- RSA (RSIP_KEY_TYPE_RSA_*) : n placed first and e placed after that.

| modulus | n | e |
|---|---|---|
| 1024 | 0 | 128 |
| 2048 | 0 | 256 |
| 3072 | 0 | 384 |
| 4096 | 0 | 512 |

**State transition**

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_AES_Cipher_Init()

fsp_err_t R_RSIP_AES_Cipher_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_aes_cipher_mode_t const *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_initial_vector* )

Starts AES cipher operation in confidentiality mode (ECB/CBC/CTR) or XTS mode.

Implements rsip_api_t::aesCipherInit.

**Conditions**

Key type of p_wrapped_key must be one of the following:

- RSIP_KEY_TYPE_AES_128, RSIP_KEY_TYPE_AES_192, RSIP_KEY_TYPE_AES_256
- RSIP_KEY_TYPE_XTS_AES_128, RSIP_KEY_TYPE_XTS_AES_256

Argument mode must be the following:

- AES key
  - RSIP_AES_CIPHER_MODE_ECB_ENC, RSIP_AES_CIPHER_MODE_ECB_DEC
  - RSIP_AES_CIPHER_MODE_CBC_ENC, RSIP_AES_CIPHER_MODE_CBC_DEC
  - RSIP_AES_CIPHER_MODE_CTR
- XTS-AES key
  - RSIP_AES_CIPHER_MODE_XTS_ENC, RSIP_AES_CIPHER_MODE_XTS_DEC

Argument p_initial_vector must be the following:

- [ECB] Not used
- [CBC] Raw initial vector
- [CTR] Raw nonce
- [XTS] Raw initial vector

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_AES |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_ARGUMENT | Input key type or mode is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_AES_Cipher_Update()

fsp_err_t R_RSIP_AES_Cipher_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint8_t *const *p_output*, uint32_t const *length* )

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::aesCipherUpdate.

**Conditions**

Argument length must be the following:

- [ECB][CBC][CTR] 0 or a multiple of 16.
- [XTS] 0 or greater than or equal to 16.

**Output length**

Output length to p_output is length.

**State transition**

This API can only be executed in **STATE_AES**, and does not cause any state transitions.

In XTS mode, if once an integer other than 0 or a multiple of 16 is input, this API can no longer be called.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | Input length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_AES_Cipher_Finish()

fsp_err_t R_RSIP_AES_Cipher_Finish ( rsip_ctrl_t *const *p_ctrl*)

Finishes AES operation.

Implements rsip_api_t::aesCipherFinish.

**State transition**

This API can only be executed in **STATE_AES**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_AES_AEAD_Init()

fsp_err_t R_RSIP_AES_AEAD_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_aes_aead_mode_t *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *nonce_length* )

Starts AES AEAD function.

Implements rsip_api_t::aesAeadInit.

**Conditions**

Key type of p_wrapped_key must be one of the following:

- RSIP_KEY_TYPE_AES_128
- RSIP_KEY_TYPE_AES_192

- RSIP_KEY_TYPE_AES_256

Argument mode accepts any member of enumeration rsip_aes_aead_mode_t.

Argument nonce_length must be the following:

- [GCM] Any length is accepted, but 12 bytes is generally recommended.
- [CCM] 7 to 13 bytes.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_AES_AEAD |
| Others | No change |

The next callable API functions in STATE_AES_AEAD are as below.

- [GCM] R_RSIP_AES_AEAD_AADUpdate(), R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_LengthsSet()

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | nonce_length is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_ARGUMENT | Input key type is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_AES_AEAD_LengthsSet()

fsp_err_t R_RSIP_AES_AEAD_LengthsSet ( rsip_ctrl_t *const *p_ctrl*, uint32_t const *total_aad_length*, uint32_t const *total_text_length*, uint32_t const *tag_length* )

Sets text and tag lengths for CCM mode.

Implements rsip_api_t::aesAeadLengthsSet.

**Conditions**

Argument total_aad_length must be equal to the length of AAD and must be 110 or less.

Argument total_test_length must be equal to the length of the plaintext or ciphertext.

Argument tag_length must be 4, 6, 8, 10, 12, 14, or 16.

**State transition**

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

The next callable API functions in STATE_AES_AEAD are as below.

- [CCM] R_RSIP_AES_AEAD_AADUpdate()

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | Input length is illegal. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

***Note***

*In GCM mode, this API must NOT be called. If called, FSP_ERR_INVALID_STATE will be returned. AAD length and text length are indeterminate and output tag length is fixed to 16 bytes.*

### ◆ R_RSIP_AES_AEAD_AADUpdate()

fsp_err_t R_RSIP_AES_AEAD_AADUpdate ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_aad*, uint32_t const *aad_length* )

Inputs Additional Authentication Data (AAD).

Implements rsip_api_t::aesAeadAadUpdate.

**State transition**

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

- [GCM] R_RSIP_AES_AEAD_AADUpdate(), R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_AADUpdate() (AAD input is not completed), R_RSIP_AES_AEAD_Update() (AAD input is completed)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | aad_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |

### ◆ R_RSIP_AES_AEAD_Update()

fsp_err_t R_RSIP_AES_AEAD_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length* )

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::aesAeadUpdate.

**Output length**

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 16 bytes.

**State transition**

This API can only be executed in **STATE_AES_AEAD**, and does not cause any state transitions.

- [GCM] R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish() (encryption), R_RSIP_AES_AEAD_Verify() (decryption)
- [CCM] R_RSIP_AES_AEAD_Update() (text input is not completed), R_RSIP_AES_AEAD_Finish() (text input is completed, encryption), R_RSIP_AES_AEAD_Verify() (text input is completed, decryption)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_SIZE | Input length is illegal. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

***Note***

*In GCM mode, if this API is skipped, GMAC will be calculated. For detailed usage, refer to example code.*

### ◆ R_RSIP_AES_AEAD_Finish()

fsp_err_t R_RSIP_AES_AEAD_Finish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t *const *p_tag* )

Finalizes an AES AEAD encryption.

Implements rsip_api_t::aesAeadFinish.

**Output length**

Output length to p_output (p_output_length) is the remaining calculated text length.

Output length to p_tag as below.

- [GCM] 16 bytes.
- [CCM] Input value as tag_length in R_RSIP_AES_AEAD_LengthsSet().

**State transition**

This API can only be executed in **STATE_AES_AEAD**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_AES_AEAD_Verify()

fsp_err_t R_RSIP_AES_AEAD_Verify ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t const *const *p_tag*, uint32_t const *tag_length* )

Finalizes an AES AEAD decryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_cipher that was input by R_RSIP_AES_GCM_DecryptUpdate(), this API will output the result of decrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

Implements rsip_api_t::aesAeadVerify.

**Conditions**

Argument tag_length must be as below.

- [GCM] 1 to 16 bytes.
- [CCM] Input value as tag_length in R_RSIP_AES_AEAD_LengthsSet().

**Output length**

Output length to p_output (p_output_length) is the remaining calculated text length.

**State transition**

This API can only be executed in **STATE_AES_AEAD**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | tag_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_AUTHENTICATION | Authentication is failed. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_AES_MAC_Init()

fsp_err_t R_RSIP_AES_MAC_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_aes_mac_mode_t const *mode*, rsip_wrapped_key_t const *const p_wrapped_key* )

Starts an AES MAC operation.

Implements rsip_api_t::aesMacInit.

**Conditions**

Key type of p_wrapped_key must be one of the following:

- RSIP_KEY_TYPE_AES_128
- RSIP_KEY_TYPE_AES_192
- RSIP_KEY_TYPE_AES_256

Argument mode accepts any member of enumeration rsip_aes_aead_mode_t.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_AES_MAC |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*To calculate AES-GMAC, please use not R_RSIP_MAC_*() but R_RSIP_AEAD_*(). For detailed usage, refer to example code.*

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_AES_MAC_Update()

fsp_err_t R_RSIP_AES_MAC_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length* )

Inputs message.

Implements rsip_api_t::aesMacUpdate.

Inside this function, the data that is input by the user is buffered until the input value of p_message exceeds 16 bytes. If the input value, p_message, is not a multiple of 16 bytes, it will be padded within the function.

**State transition**

This API can only be executed in **STATE_AES_MAC**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_AES_MAC_SignFinish()

fsp_err_t R_RSIP_AES_MAC_SignFinish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_mac* )

Outputs AES MAC.

Implements rsip_api_t::aesMacSignFinish.

**Output length**

Output length to p_mac is 16 bytes.

**State transition**

This API can only be executed in **STATE_AES_MAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_AES_MAC_VerifyFinish()

fsp_err_t R_RSIP_AES_MAC_VerifyFinish ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*, uint32_t const *mac_length* )

Verifies AES MAC.

Implements rsip_api_t::aesMacVerifyFinish.

**Conditions**

Argument mac_length must be as below.

- [CMAC] 2 to 16 bytes.

**State transition**

This API can only be executed in **STATE_AES_MAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | mac_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Internal error. |
| FSP_ERR_CRYPTO_RSIP_AUTHENTICATION | Authentication is failed. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Init()

fsp_err_t R_RSIP_ChaCha20_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *counter* )

Prepares a ChaCha20 function.

Implements rsip_api_t::chacha20Init.

**Conditions**

Key type of p_wrapped_key must be RSIP_KEY_TYPE_CHACHA20.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_CHACHA |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Update()

fsp_err_t R_RSIP_ChaCha20_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length* )

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::chacha20Update.

**Output length**

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 64 bytes.

**State transition**

This API can only be executed in **RSIP_STATE_CHACHA**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_ChaCha20_Finish()

fsp_err_t R_RSIP_ChaCha20_Finish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length* )

Finalize ChaCha20 operation.

Implements rsip_api_t::chacha20Finish.

**Output length**

Output length to p_output (p_output_length) is the remaining calculated text length.

**State transition**

This API can only be executed in **RSIP_STATE_CHACHA**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Poly1305_Init()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_chacha_poly_mode_t const *mode*, rsip_wrapped_key_t const *const *p_wrapped_key*, uint8_t const *const *p_nonce*, uint32_t const *nonce_length* )

Prepares a ChaCha20-Poly1305 function.

Implements rsip_api_t::chacha20Poly1305Init.

**Conditions**

Key type of p_wrapped_key must be RSIP_KEY_TYPE_CHACHA20.

Argument mode accepts any member of enumeration rsip_chacha_poly_mode_t.

Argument nonce_length must be 12 bytes.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_CHACHA_POLY |
| Others | No change |

The next callable API functions in STATE_CHACHA_POLY are as below.

- R_RSIP_ChaCha20_Poly1305_AADUpdate(), R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | nonce_length is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_ARGUMENT | Input mode is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Poly1305_AADUpdate()

fsp_err_t R_RSIP_ChaCha20_Poly1305_AADUpdate ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_aad*, uint32_t const *aad_length* )

Inputs Additional Authentication Data (AAD).

Implements rsip_api_t::chacha20Poly1305AadUpdate.

**State transition**

This API can only be executed in **STATE_CHACHA_POLY**, and does not cause any state transitions.

- R_RSIP_ChaCha20_Poly1305_AADUpdate(), R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_ChaCha20_Poly1305_Update()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_input*, uint32_t const *input_length*, uint8_t *const *p_output*, uint32_t *const *p_output_length* )

Encrypts plaintext or decrypts ciphertext.

Implements rsip_api_t::chacha20Poly1305Update.

**Conditions**

At least one byte of data must be input with this API before calling R_RSIP_ChaCha20_Poly1305_Finish() or R_RSIP_ChaCha20_Poly1305_Verify().

**Output length**

Output length to p_output (p_output_length) is calculated text that has not yet been output in multiple of 64 bytes.

**State transition**

This API can only be executed in **STATE_CHACHA_POLY**, and does not cause any state transitions.

- R_RSIP_ChaCha20_Poly1305_Update(), R_RSIP_ChaCha20_Poly1305_Finish() (encryption), R_RSIP_ChaCha20_Poly1305_Verify() (decryption)

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Poly1305_Finish()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Finish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t *const *p_tag* )

Finalizes a ChaCha20-Poly1305 encryption.

Implements rsip_api_t::chacha20Poly1305Finish.

**Output length**

Output length to p_output (p_output_length) is the remaining calculated text length.

Output length to p_tag is 16 bytes.

**State transition**

This API can only be executed in **STATE_CHACHA_POLY**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| FSP_ERR_INVALID_SIZE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | Input text length is illegal. (zero byte) |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ChaCha20_Poly1305_Verify()

fsp_err_t R_RSIP_ChaCha20_Poly1305_Verify ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_output*, uint32_t *const *p_output_length*, uint8_t const *const *p_tag*, uint32_t const *tag_length* )

Finalizes a ChaCha20-Poly1305 decryption.

Implements rsip_api_t::chacha20Poly1305Verify.

**Conditions**

Argument tag_length must be 16 bytes.

**Output length**

Output length to p_output (p_output_length) is the remaining calculated text length.

**State transition**

This API can only be executed in **STATE_CHACHA_POLY**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | tag_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_AUTHENTICATION | Authentication is failed. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ECDSA_Sign()

fsp_err_t R_RSIP_ECDSA_Sign ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_hash*, uint8_t *const *p_signature* )

Generates an ECDSA signature.

Implements rsip_api_t::ecdsaSign.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE
- RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE
- RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE
- RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE
- RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

| Data Format for secp521r1 (132 byte) | | | |
|---|---|---|---|
| zero padding (7 bit) | signature r (521 bit) | zero padding (7 bit) | signature s (521 bit) |

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | • Input parameter is illegal.<br>• Signature generation is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_ECDSA_Verify()

fsp_err_t R_RSIP_ECDSA_Verify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies an ECDSA signature.

Implements rsip_api_t::ecdsaVerify.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

| Data Format for secp521r1 (132 byte) | | | |
|---|---|---|---|
| zero padding (7 bit) | signature r (521 bit) | zero padding (7 bit) | signature s (521 bit) |

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | • Input parameter is illegal.<br>• Signature verification is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_PKI_ECDSA_CertVerify()

fsp_err_t R_RSIP_PKI_ECDSA_CertVerify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies a public key certificate with ECDSA.

Implements rsip_api_t::pkiEcdsaCertVerify.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length.

For secp521r1 operation, the length of p_hash must be set to 64 bytes.

For secp521r1 operation, the length of the argument p_signature must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

| Data Format for secp521r1 (132 byte) | | | |
|---|---|---|---|
| zero padding (7 bit) | signature r (521 bit) | zero padding (7 bit) | signature s (521 bit) |

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | • Input parameter is illegal.<br>• Signature verification is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_PureEdDSA_Sign()

fsp_err_t R_RSIP_PureEdDSA_Sign ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_message*, uint64_t const *message_length*, uint8_t *const *p_signature* )

Generates an EdDSA signature.

Implements rsip_api_t::eddsaSign.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_SIZE | Input length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_PureEdDSA_Verify()

fsp_err_t R_RSIP_PureEdDSA_Verify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_message*, uint64_t const *message_length*, uint8_t const *const *p_signature* )

Verifies an EdDSA signature.

Implements rsip_api_t::eddsaVerify.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_INVALID_SIZE | Input length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_ECDH_KeyAgree()

fsp_err_t R_RSIP_ECDH_KeyAgree ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_wrapped_secret_t *const *p_wrapped_secret* )

Computes ECDH secret with wrapped private key and wrapped public key.

Implements rsip_api_t::ecdhKeyAgree.

**Conditions**

Key type of p_wrapped_private_key and p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_ECDH_PlainKeyAgree()

fsp_err_t R_RSIP_ECDH_PlainKeyAgree ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_plain_public_key*, rsip_wrapped_secret_t *const *p_wrapped_secret* )

Computes ECDH secret with wrapped private key and plain public key.

Implements rsip_api_t::ecdhPlainKeyAgree.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC

For secp521r1 operation, the length of the argument p_plain_public_key must be set as 132 byte. Since 521 bit is not a 8-bit multiple, zero padding is required and the data format is as follows:

| Data Format for secp521r1 (132 byte) | | | |
|---|---|---|---|
| zero padding (7 bit) | public_key Qx (521 bit) | zero padding (7 bit) | public_key Qy (521 bit) |

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**Attention**

Unauthenticated public key of Diffie–Hellman key exchange generally carry the risk of man-in-the-middle (MITM) attack. Please use R_RSIP_ECDH_KeyAgree() or implement signature verification whenever possible.

### ◆ R_RSIP_KDF_SHA_Init()

fsp_err_t R_RSIP_KDF_SHA_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type* )

Prepares a SHA generation.

Implements rsip_api_t::kdfShaInit.

**Conditions**

Argument hash_type must be one of the following:
- RSIP_HASH_TYPE_SHA256
- RSIP_HASH_TYPE_SHA384

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_KDF_SHA |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |

### ◆ R_RSIP_KDF_SHA_ECDHSecretUpdate()

fsp_err_t R_RSIP_KDF_SHA_ECDHSecretUpdate ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret* )

Inputs wrapped ECDH secret as a message. Input the messages required to perform the SHA operation using this API and R_RSIP_KDF_SHA_Update() in the desired order.

Implements rsip_api_t::kdfShaEcdhSecretUpdate.

**Conditions**

The argument p_wrapped_secret must be input the result of R_RSIP_ECDH_KeyAgree() or R_RSIP_ECDH_PlainKeyAgree(). This API can only be called if there is no previous message input, or if a 4-byte message has been input using R_RSIP_KDF_SHA_Update(). This API cannot be called multiple times in a process of performing SHA operation.

**State transition**

This API can only be executed in **STATE_KDF_SHA**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_SHA_Update()

fsp_err_t R_RSIP_KDF_SHA_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length* )

Inputs message. Input the messages required to perform the SHA operation using this API and R_RSIP_KDF_SHA_ECDHSecretUpdate() in the desired order.

Implements rsip_api_t::kdfShaUpdate.

**State transition**

This API can only be executed in **STATE_KDF_SHA**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | Input message_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*For RSIP-E11A, the total message length that can be input is the block size of the SHA algorithm used.*

### ◆ R_RSIP_KDF_SHA_Finish()

fsp_err_t R_RSIP_KDF_SHA_Finish ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t *const *p_wrapped_dkm* )

Finalizes a SHA operation and generate DKM (Derived Keying Material).

Implements rsip_api_t::kdfShaFinish.

**State transition**

This API can only be executed in **STATE_KDF_SHA**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_SHA_Suspend()

fsp_err_t R_RSIP_KDF_SHA_Suspend ( rsip_ctrl_t *const *p_ctrl*, rsip_kdf_sha_handle_t *const *p_handle* )

Suspends SHA operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::kdfShaSuspend.

**State transition**

This API can only be executed in **STATE_KDF_SHA**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_SHA_Resume()

fsp_err_t R_RSIP_KDF_SHA_Resume ( rsip_ctrl_t *const *p_ctrl*, rsip_kdf_sha_handle_t const *const *p_handle* )

Resumes SHA operation suspended by R_RSIP_KDF_SHA_Suspend().

Implements rsip_api_t::kdfShaResume.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_KDF_SHA |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_KDF_HMAC_DKMKeyImport()

fsp_err_t R_RSIP_KDF_HMAC_DKMKeyImport ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, uint32_t const *key_length*, rsip_wrapped_key_t *const *p_wrapped_key* )

Converts wrapped MAC to wrapped HMAC key for KDF.

Implements rsip_api_t::kdfHmacDkmKeyImport.

**Conditions**

Argument key_type must be one of the following:

- RSIP_KEY_TYPE_KDF_HMAC_SHA256
- RSIP_KEY_TYPE_KDF_HMAC_SHA384
- RSIP_KEY_TYPE_KDF_HMAC_SHA512

The argument p_wrapped_mac must be input the result of R_RSIP_KDF_HMAC_SignFinish().

Argument kdf_data_length depends on key type.

- 32 to 64 ( RSIP_KEY_TYPE_KDF_HMAC_SHA256)
- 48 to 64 ( RSIP_KEY_TYPE_KDF_HMAC_SHA384)
- 64 ( RSIP_KEY_TYPE_KDF_HMAC_SHA512)

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_ECDHSecretKeyImport()

fsp_err_t R_RSIP_KDF_HMAC_ECDHSecretKeyImport ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret*, rsip_wrapped_key_t *const *p_wrapped_key* )

Converts wrapped ECDH secret to wrapped HMAC key for KDF.

Implements rsip_api_t::kdfHmacEcdhSecretKeyImport.

**Conditions**

Argument key_type must be one of the following:

- RSIP_KEY_TYPE_KDF_HMAC_SHA256
- RSIP_KEY_TYPE_KDF_HMAC_SHA384
- RSIP_KEY_TYPE_KDF_HMAC_SHA512

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_Init()

fsp_err_t R_RSIP_KDF_HMAC_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key* )

Prepares a HMAC generation.

Implements rsip_api_t::kdfHmacInit.

**Conditions**

Argument p_wrapped_key must be one of the following:
- Output data p_wrapped_key from R_RSIP_KDF_ECDHSecretKeyImport().
- Key type of p_wrapped_key is RSIP_KEY_TYPE_HMAC_*.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_KDF_HMAC |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |

### ◆ R_RSIP_KDF_HMAC_DKMUpdate()

fsp_err_t R_RSIP_KDF_HMAC_DKMUpdate ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm* )

Inputs wrapped Derived Keying Material (DKM) as a message.

Implements rsip_api_t::kdfHmacDkmUpdate.

**Conditions**

The argument p_wrapped_mac must be input the result of R_RSIP_KDF_HMAC_SignFinish().

**State transition**

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_KDF_HMAC_ECDHSecretUpdate()

fsp_err_t R_RSIP_KDF_HMAC_ECDHSecretUpdate ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_secret_t const *const *p_wrapped_secret* )

Inputs wrapped ECDH secret as a message.

Implements rsip_api_t::kdfHmacEcdhSecretUpdate.

**Conditions**

The argument p_wrapped_secret must be input the result of R_RSIP_ECDH_KeyAgree() or R_RSIP_ECDH_PlainKeyAgree().

**State transition**

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_Update()

fsp_err_t R_RSIP_KDF_HMAC_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length* )

Inputs message.

Implements rsip_api_t::kdfHmacUpdate.

**State transition**

This API can only be executed in **STATE_KDF_HMAC**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_SignFinish()

fsp_err_t R_RSIP_KDF_HMAC_SignFinish ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t *const *p_wrapped_dkm* )

Finalizes a HMAC generation.

Implements rsip_api_t::kdfHmacSignFinish.

**State transition**

This API can only be executed in **STATE_KDF_HMAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_Suspend()

fsp_err_t R_RSIP_KDF_HMAC_Suspend ( rsip_ctrl_t *const *p_ctrl*, rsip_kdf_hmac_handle_t *const *p_handle* )

Suspends HMAC operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::kdfHmacSuspend.

**State transition**

This API can only be executed in **STATE_KDF_HMAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_HMAC_Resume()

fsp_err_t R_RSIP_KDF_HMAC_Resume ( rsip_ctrl_t *const *p_ctrl*, rsip_kdf_hmac_handle_t const *const *p_handle* )

Resumes HMAC operation suspended by R_RSIP_KDF_HMAC_Suspend().

Implements rsip_api_t::kdfHmacResume.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_KDF_HMAC |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_KDF_DKMConcatenate()

fsp_err_t R_RSIP_KDF_DKMConcatenate ( rsip_wrapped_dkm_t *const *p_wrapped_dkm1*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm2*, uint32_t const *wrapped_dkm1_buffer_length* )

Concatenates two wrapped derived keying materials (DKMs).

DKM1 || DKM2 is output to p_wrapped_dkm1.

Implements rsip_api_t::kdfDkmConcatenate.

**State transition**

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |

### ◆ R_RSIP_KDF_DerivedKeyImport()

fsp_err_t R_RSIP_KDF_DerivedKeyImport ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, uint32_t const *position*, rsip_wrapped_key_t *const *p_wrapped_key* )

Outputs a wrapped key from KDF output.

Implements rsip_api_t::kdfDerivedKeyImport.

**Conditions**

Argument p_wrapped_mac must be input the result of R_RSIP_KDF_HMAC_SignFinish() or R_RSIP_KDF_MACConcatenate(). Argument key_type must be one of the following:

- RSIP_KEY_TYPE_AES_128
- RSIP_KEY_TYPE_AES_256
- RSIP_KEY_TYPE_HMAC_SHA256
- RSIP_KEY_TYPE_HMAC_SHA384
- RSIP_KEY_TYPE_HMAC_SHA512

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_KDF_DerivedIVWrap()

fsp_err_t R_RSIP_KDF_DerivedIVWrap ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_dkm_t const *const *p_wrapped_dkm*, rsip_initial_vector_type_t const *initial_vector_type*, uint32_t const *position*, uint8_t const *const *p_tls_sequence_num*, uint8_t *const *p_wrapped_initial_vector* )

Outputs a initial vector from KDF output.

Implements rsip_api_t::kdfDerivedIvWrap

**Conditions**

Argument p_wrapped_mac must be input the result of R_RSIP_KDF_HMAC_SignFinish() or R_RSIP_KDF_MACConcatenate(). Argument initial_vector_type must be one of the following:

- RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_OTF_Init()

fsp_err_t R_RSIP_OTF_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_otf_channel_t const *channel*, rsip_wrapped_key_t *const *p_wrapped_key*, uint8_t const *const *p_seed* )

Initialize on-the-fly decryption on RSIP. Implements rsip_api_t::otfInit.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled or selected channel is invalid. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms

**Attention**

This function is only part of on-the-fly decryption activation process and intended to be called from a higher level FSP module. Even if a user calls this function directly, the feature will not be enabled.

### ◆ R_RSIP_PKI_VerifiedCertInfoExport()

fsp_err_t R_RSIP_PKI_VerifiedCertInfoExport ( rsip_ctrl_t *const *p_ctrl*, rsip_verified_cert_info_t *const *p_verified_cert_info* )

Exports verified certificate information stored in this driver.

The certificate is last called function R_RSIP_PKI_ECDSA_CertVerify(), R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify(), R_RSIP_PKI_RSASSA_PSS_CertVerify(), or R_RSIP_PKI_VerifiedCertInfoImport().

Implements rsip_api_t::pkiVerifiedCertInfoExport.

**State transition**

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |

### ◆ R_RSIP_PKI_VerifiedCertInfoImport()

fsp_err_t R_RSIP_PKI_VerifiedCertInfoImport ( rsip_ctrl_t *const *p_ctrl*, rsip_verified_cert_info_t const *const *p_verified_cert_info* )

Imports verified certificate information.

Implements rsip_api_t::pkiVerifiedCertInfoImport.

**State transition**

This API can be executed in **any state** including STATE_INITIAL, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |

### ◆ R_RSIP_PKI_CertKeyImport()

fsp_err_t R_RSIP_PKI_CertKeyImport ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_cert*, uint32_t const *cert_length*, uint8_t const *const *p_key_param1*, uint32_t const *key_param1_length*, uint8_t const *const *p_key_param2*, uint32_t const *key_param2_length*, rsip_hash_type_t const *hash_function*, rsip_wrapped_key_t *const *p_wrapped_public_key* )

Wraps the public key in the verified public key certificate.

Implements rsip_api_t::pkiCertKeyImport.

**Conditions**

Argument key_type must be one of the following:

- RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC
- RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC
- RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

- *Argument p_cert must correspond to the signature of the last called function R_RSIP_PKI_ECDSA_CertVerify(), R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify(), or R_RSIP_PKI_RSASSA_PSS_CertVerify().*
- *For arguments p_key_param1 and p_key_param2, specify an address inside the certificate. Addresses outside the certificate are invalid.*

**See also**

Section Supported Algorithms

### ◆ R_RSIP_RSA_Encrypt()

fsp_err_t R_RSIP_RSA_Encrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_plain*, uint8_t *const *p_cipher* )

Encrypts plaintext with raw RSA.

Implements rsip_api_t::rsaEncrypt.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:
- RSIP_KEY_TYPE_RSA_2048_PUBLIC
- RSIP_KEY_TYPE_RSA_3072_PUBLIC
- RSIP_KEY_TYPE_RSA_4096_PUBLIC

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*This API provides RSA low-level primitives (RSAEP/RSAVP1). It should be used in conjunction with any padding scheme.*

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSA_Decrypt()

fsp_err_t R_RSIP_RSA_Decrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain* )

Decrypts ciphertext with raw RSA.

Implements rsip_api_t::rsaDecrypt.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:
- RSIP_KEY_TYPE_RSA_2048_PRIVATE
- RSIP_KEY_TYPE_RSA_3072_PRIVATE
- RSIP_KEY_TYPE_RSA_4096_PRIVATE

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*This API provides RSA low-level primitives (RSADP/RSASP1). It should be used in conjunction with any padding scheme.*

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSAES_PKCS1_V1_5_Encrypt()

fsp_err_t R_RSIP_RSAES_PKCS1_V1_5_Encrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, uint8_t const *const *p_plain*, uint32_t const *plain_length*, uint8_t *const *p_cipher* )

Encrypts plaintext with RSAES-PKCS1-v1_5.

Implements rsip_api_t::rsaesPkcs1V15Encrypt.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PUBLIC
- RSIP_KEY_TYPE_RSA_3072_PUBLIC
- RSIP_KEY_TYPE_RSA_4096_PUBLIC

mLen (plain_length) and k (modulus length) must meet the following condition.

- mLen <= k - 11

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSAES_PKCS1_V1_5_Decrypt()

fsp_err_t R_RSIP_RSAES_PKCS1_V1_5_Decrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain*, uint32_t *const *p_plain_length*, uint32_t const *plain_buffer_length* )

Decrypts with RSAES-PKCS1-v1_5.

Implements rsip_api_t::rsaesPkcs1V15Decrypt.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PRIVATE
- RSIP_KEY_TYPE_RSA_3072_PRIVATE
- RSIP_KEY_TYPE_RSA_4096_PRIVATE

plain_buffer_length must be greater than or equal to mLen(plaintext length).

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*This API skips the ciphertext length checking at RFC8017 (PKCS#1 v2.2) Section 7.2.2 Step 1.*

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSAES_OAEP_Encrypt()

fsp_err_t R_RSIP_RSAES_OAEP_Encrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint8_t const *const *p_label*, uint32_t const *label_length*, uint8_t const *const *p_plain*, uint32_t const *plain_length*, uint8_t *const *p_cipher* )

Encrypts plaintext with RSAES-OAEP.

Implements rsip_api_t::rsaesOaepEncrypt.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PUBLIC
- RSIP_KEY_TYPE_RSA_3072_PUBLIC
- RSIP_KEY_TYPE_RSA_4096_PUBLIC

mLen (plain_length), hLen (hash length of hash_function), and k (modulus length) must meet the following condition.

- mLen <= k - 2 hLen - 2

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Argument mask_generation_function accepts any member of enumeration rsip_mgf_type_t.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSAES_OAEP_Decrypt()

fsp_err_t R_RSIP_RSAES_OAEP_Decrypt ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint8_t const *const *p_label*, uint32_t const *label_length*, uint8_t const *const *p_cipher*, uint8_t *const *p_plain*, uint32_t *const *p_plain_length*, uint32_t const *plain_buffer_length* )

Decrypts ciphertext with RSAES-OAEP.

Implements rsip_api_t::rsaesOaepDecrypt.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PRIVATE
- RSIP_KEY_TYPE_RSA_3072_PRIVATE
- RSIP_KEY_TYPE_RSA_4096_PRIVATE

hLen (hash length of hash_function) and k (modulus length) must meet the following condition.

- k >= 2 hLen + 2

plain_buffer_length must be greater than or equal to mLen(plaintext length).

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Argument mask_generation_function accepts any member of enumeration rsip_mgf_type_t.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

***Note***

*This API skips the ciphertext length checking at RFC8017 (PKCS#1 v2.2) Section 7.1.2 Step 1.*

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSASSA_PKCS1_V1_5_Sign()

fsp_err_t R_RSIP_RSASSA_PKCS1_V1_5_Sign ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t *const *p_signature* )

Signs message with RSASSA-PKCS1-v1_5.

Implements rsip_api_t::rsassaPkcs1V15Sign.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PRIVATE
- RSIP_KEY_TYPE_RSA_3072_PRIVATE
- RSIP_KEY_TYPE_RSA_4096_PRIVATE

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Message hash p_hash should be computed in advance with hash_function.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSASSA_PKCS1_V1_5_Verify()

fsp_err_t R_RSIP_RSASSA_PKCS1_V1_5_Verify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies signature with RSASSA-PKCS1-v1_5.

Implements rsip_api_t::rsassaPkcs1V15Verify.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PUBLIC
- RSIP_KEY_TYPE_RSA_3072_PUBLIC
- RSIP_KEY_TYPE_RSA_4096_PUBLIC

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Message hash p_hash should be computed in advance with hash_function.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSASSA_PSS_Sign()

fsp_err_t R_RSIP_RSASSA_PSS_Sign ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_private_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t *const

*p_signature* )

Signs message with RSASSA-PSS.

Implements rsip_api_t::rsassaPssSign.

**Conditions**

Key type of p_wrapped_private_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PRIVATE
- RSIP_KEY_TYPE_RSA_3072_PRIVATE
- RSIP_KEY_TYPE_RSA_4096_PRIVATE

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Argument mask_generation_function accepts any member of enumeration rsip_mgf_type_t.

Message hash p_hash should be computed in advance with hash_function.

Salt length salt_length must be one of the following:

- Any member of enumeration rsip_rsa_salt_length_t
  - RSIP_RSA_SALT_LENGTH_AUTO
  - RSIP_RSA_SALT_LENGTH_HASH
  - RSIP_RSA_SALT_LENGTH_MAX
- Integers that satisfies the formula: sLen <= emLen - hLen - 2
  - sLen is salt_length
  - emLen is the same as modulus length
  - hLen is the hash length

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_RSASSA_PSS_Verify()

fsp_err_t R_RSIP_RSASSA_PSS_Verify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies signature with RSASSA-PSS. Implements rsip_api_t::rsassaPssVerify.

**Conditions**

Key type of p_wrapped_public_key must be one of the following:

- RSIP_KEY_TYPE_RSA_2048_PUBLIC
- RSIP_KEY_TYPE_RSA_3072_PUBLIC
- RSIP_KEY_TYPE_RSA_4096_PUBLIC

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

Argument mask_generation_function accepts any member of enumeration rsip_mgf_type_t.

Message hash p_hash should be computed in advance with hash_function.

Salt length salt_length must be one of the following:

- Any member of enumeration rsip_rsa_salt_length_t
  - RSIP_RSA_SALT_LENGTH_AUTO
  - RSIP_RSA_SALT_LENGTH_HASH
  - RSIP_RSA_SALT_LENGTH_MAX
- Integers that satisfies the formula: sLen <= emLen - hLen - 2
  - sLen is salt_length
  - emLen is the same as modulus length
  - hLen is the hash length

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| | |

| | |
|---|---|
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify()

fsp_err_t R_RSIP_PKI_RSASSA_PKCS1_V1_5_CertVerify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies a certificate with RSASSA-PKCS1-v1_5.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | hash_function | Hash function in EMSA-PKCS1-v1_5. |
| [in] | p_hash | Pointer to input hash. |
| [in] | p_signature | Pointer to input signature. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_PKI_RSASSA_PSS_CertVerify()

fsp_err_t R_RSIP_PKI_RSASSA_PSS_CertVerify ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_public_key*, rsip_hash_type_t const *hash_function*, rsip_mgf_type_t const *mask_generation_function*, uint32_t const *salt_length*, uint8_t const *const *p_hash*, uint8_t const *const *p_signature* )

Verifies a certificate with RSASSA-PSS.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | hash_function | Hash function in EMSA-PSS-VERIFY. |
| [in] | mask_generation_function | Mask generation function in EMSA-PSS-VERIFY. |
| [in] | salt_length | Salt length. RSIP_RSA_SALT_LENGTH_AUTO, RSIP_RSA_SALT_LENGTH_HASH, RSIP_RSA_SALT_LENGTH_MAX, 0, or positive integers can be set, where salt_length <= emLen - hLen - 2 (emLen is the same as the key length and hLen is the hash length). |
| [in] | p_hash | Pointer to input hash. |
| [in] | p_signature | Pointer to input signature. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | Input parameter is invalid. |
| FSP_ERR_INVALID_SIZE | Any length is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLI | A resource conflict occurred because a |

| | |
|---|---|
| CT | hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_SHA_Compute()

fsp_err_t R_RSIP_SHA_Compute ( rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t *const *p_digest* )

Generates SHA message digest. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::shaCompute.

**Conditions**

See R_RSIP_SHA_Init().

**Output length**

See R_RSIP_SHA_Finish().

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_SHA_Init()

fsp_err_t R_RSIP_SHA_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_hash_type_t const *hash_type* )

Starts SHA operation.

Implements rsip_api_t::shaInit.

**Conditions**

Argument hash_function accepts any member of enumeration rsip_hash_type_t.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_SHA |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_SHA_Update()

fsp_err_t R_RSIP_SHA_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length* )

Inputs SHA message. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::shaUpdate.

**State transition**

This API can only be executed in **STATE_SHA**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_SHA_Finish()

fsp_err_t R_RSIP_SHA_Finish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_digest* )

Outputs SHA message digest.

Implements rsip_api_t::shaFinish.

**Output length**

Output length to p_digest depends on hash_function.

- 28 ( RSIP_HASH_TYPE_SHA224, RSIP_HASH_TYPE_SHA512_224, RSIP_HASH_TYPE_SHA3_224)
- 32 ( RSIP_HASH_TYPE_SHA256, RSIP_HASH_TYPE_SHA512_256, RSIP_HASH_TYPE_SHA3_256)
- 48 ( RSIP_HASH_TYPE_SHA384, RSIP_HASH_TYPE_SHA3_384)
- 64 ( RSIP_HASH_TYPE_SHA512, RSIP_HASH_TYPE_SHA3_512)

**State transition**

This API can only be executed in **STATE_SHA**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_SHA_Suspend()

fsp_err_t R_RSIP_SHA_Suspend ( rsip_ctrl_t *const *p_ctrl*, rsip_sha_handle_t *const *p_handle* )

Suspends SHA operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::shaSuspend.

**State transition**

This API can only be executed in **STATE_SHA**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_SHA_Resume()

fsp_err_t R_RSIP_SHA_Resume ( rsip_ctrl_t *const *p_ctrl*, rsip_sha_handle_t const *const *p_handle* )

Resumes SHA operation suspended by R_RSIP_SHA_Suspend().

Implements rsip_api_t::shaResume.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_SHA |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_HMAC_Compute()

fsp_err_t R_RSIP_HMAC_Compute ( rsip_ctrl_t *const *p_ctrl*, const rsip_wrapped_key_t * *p_wrapped_key*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t *const *p_mac* )

Generates HMAC. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacCompute.

**Conditions**

See R_RSIP_HMAC_Init().

**Output length**

See R_RSIP_HMAC_SignFinish().

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_HMAC_Verify()

fsp_err_t R_RSIP_HMAC_Verify ( rsip_ctrl_t *const *p_ctrl*, const rsip_wrapped_key_t * *p_wrapped_key*, uint8_t const *const *p_message*, uint32_t const *message_length*, uint8_t const *const *p_mac*, uint32_t const *mac_length* )

Verifies HMAC. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacVerify.

**Conditions**

See R_RSIP_HMAC_Init() and R_RSIP_HMAC_VerifyFinish().

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | mac_length is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key value is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | MAC verification is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_HMAC_Init()

fsp_err_t R_RSIP_HMAC_Init ( rsip_ctrl_t *const *p_ctrl*, rsip_wrapped_key_t const *const *p_wrapped_key* )

Starts HMAC operation.

Implements rsip_api_t::hmacInit.

**Conditions**

Key type of p_wrapped_key must be one of the following:

- RSIP_KEY_TYPE_HMAC_SHA224
- RSIP_KEY_TYPE_HMAC_SHA256
- RSIP_KEY_TYPE_HMAC_SHA384
- RSIP_KEY_TYPE_HMAC_SHA512
- RSIP_KEY_TYPE_HMAC_SHA512_224
- RSIP_KEY_TYPE_HMAC_SHA512_256

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_HMAC |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_NOT_ENABLED | Input key type is disabled in this function by configuration. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |

**See also**

Section Supported Algorithms.

### ◆ R_RSIP_HMAC_Update()

fsp_err_t R_RSIP_HMAC_Update ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_message*, uint32_t const *message_length* )

Inputs HMAC message. (Total input message must be less than 2^64 bits.)

Implements rsip_api_t::hmacUpdate.

**State transition**

This API can only be executed in **STATE_HMAC**, and does not cause any state transitions.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_HMAC_SignFinish()

fsp_err_t R_RSIP_HMAC_SignFinish ( rsip_ctrl_t *const *p_ctrl*, uint8_t *const *p_mac* )

Outputs HMAC.

Implements rsip_api_t::hmacSignFinish.

**Output length**

Output length to p_mac depends on key type of p_wrapped_key.

- 28 ( RSIP_KEY_TYPE_HMAC_SHA224, RSIP_KEY_TYPE_HMAC_SHA512_224)
- 32 ( RSIP_KEY_TYPE_HMAC_SHA256, RSIP_KEY_TYPE_HMAC_SHA512_256)
- 48 ( RSIP_KEY_TYPE_HMAC_SHA384)
- 64 ( RSIP_KEY_TYPE_HMAC_SHA512)

**State transition**

This API can only be executed in **STATE_HMAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_HMAC_VerifyFinish()

fsp_err_t R_RSIP_HMAC_VerifyFinish ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac*, uint32_t const *mac_length* )

Verifies HMAC.

Implements rsip_api_t::hmacVerifyFinish.

**Conditions**

Argument mac_length depends on key type of p_wrapped_key. Usually the longest length is recommended.

- 4 to 28 ( RSIP_KEY_TYPE_HMAC_SHA224, RSIP_KEY_TYPE_HMAC_SHA512_224)
- 4 to 32 ( RSIP_KEY_TYPE_HMAC_SHA256, RSIP_KEY_TYPE_HMAC_SHA512_256)
- 4 to 48 ( RSIP_KEY_TYPE_HMAC_SHA384)
- 4 to 64 ( RSIP_KEY_TYPE_HMAC_SHA512)

**State transition**

This API can only be executed in **STATE_HMAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| FSP_ERR_ASSERTION | No change |
| FSP_ERR_NOT_OPEN | No change |
| FSP_ERR_INVALID_STATE | No change |
| FSP_ERR_INVALID_SIZE | No change |
| Others | STATE_MAIN |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_INVALID_SIZE | mac_length is illegal. |
| FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL | Input key is illegal. |
| FSP_ERR_CRYPTO_RSIP_FAIL | MAC verification is failed. |
| FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_HMAC_Suspend()

fsp_err_t R_RSIP_HMAC_Suspend ( rsip_ctrl_t *const *p_ctrl*, rsip_hmac_handle_t *const *p_handle* )

Suspends HMAC operation.

This API releases RSIP resource and outputs intermediate results. Therefore, it can be used in the following cases:

- Execute another cryptographic operations during inputting successive chunks of the message.
- Reuse intermediate results.

Implements rsip_api_t::hmacSuspend.

**State transition**

This API can only be executed in **STATE_HMAC**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_MAIN |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_CRYPTO_RSIP_FATAL | Software corruption is detected. |

### ◆ R_RSIP_HMAC_Resume()

fsp_err_t R_RSIP_HMAC_Resume ( rsip_ctrl_t *const *p_ctrl*, rsip_hmac_handle_t const *const *p_handle* )

Resumes HMAC operation suspended by R_RSIP_HMAC_Suspend().

Implements rsip_api_t::hmacResume.

**State transition**

This API can only be executed in **STATE_MAIN**, and causes state transition.

| Return value | Next state |
|---|---|
| FSP_SUCCESS | STATE_HMAC |
| Others | No change |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |

### ◆ R_RSIP_FSBL_OEM_BL_Digest_Generate()

fsp_err_t R_RSIP_FSBL_OEM_BL_Digest_Generate ( rsip_ctrl_t *const *p_ctrl*, uint8_t const *const *p_key_cert*, uint32_t const *key_cert_max_length*, uint8_t const *const *p_code_cert*, uint32_t const *code_cert_max_length*, uint32_t *const *p_mac* )

Verifies a Manifest signature and outputs the MAC of image and code certificate. The hash algorithm only supports HMAC-SHA256. The signature algorithm only supports ECDSA secp256r1.

**State transition**

This API can only be executed in **STATE_MAIN**, and does not cause any state transitions.

***Note***

*If this API returns FSP_ERR_SB_LOWER_IMAGE_VERSION or FSP_ERR_SB_CRYPTO_AUTH_FAIL, RSIP stops working. In this case, RSIP can be resumed by executing R_RSIP_Close() and R_RSIP_Open() in that order.*

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_key_cert | Pointer to key certificate. |
| [in] | key_cert_max_length | Maximum length of key certificate. |
| [in] | p_code_cert | Pointer to key code certificate. |

| | | |
|---|---|---|
| [in] | code_cert_max_length | Maximum length of code certificate. |
| [out] | p_mac | Pointer to output MAC. It is output in TLV format and consists of a 4-byte Type-Length and a 32-byte Value. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_STATE | Internal state is illegal. |
| FSP_ERR_SB_INTERNAL_FAIL | An internal failure. |
| FSP_ERR_SB_INVALID_ARG | An invalid argument was entered. |
| FSP_ERR_SB_UNSUPPORTED_FUNCTION | Unsupported function executed. |
| FSP_ERR_SB_INVALID_ALIGNMENT | Data entered with incorrect alignment. |
| FSP_ERR_SB_SAME_IMAGE_VERSION | An image of the same version as the current version is input. (verification completed successfully) |
| FSP_ERR_SB_LOWER_IMAGE_VERSION | Image version lower than the current image version is installed. |
| FSP_ERR_SB_MANI_INVALID_MAGIC | An invalid magic number is set. |
| FSP_ERR_SB_MANI_UNSUPPORTED_VERSION | Unsupported version is set. |
| FSP_ERR_SB_MANI_OUT_OF_RANGE_LEN | Out of range TLV Length is set. |
| FSP_ERR_SB_MANI_TLV_FIELD_ERR | Missing required TLV field. |
| FSP_ERR_SB_MANI_TLV_INVALID_LEN | The length exceeding the end of the manifest is specified in length of the TLV field. |
| FSP_ERR_SB_MANI_INVALID_IMAGE_LEN | An invalid image length is set. |
| FSP_ERR_SB_MANI_MISMATCH_SIGN_ALGORITHM | There is a wrong combination of signature algorithms. |
| FSP_ERR_SB_MANI_UNSUPPORTED_ALGORITHM | An algorithm was specified that the manifest does not support. |
| FSP_ERR_SB_CRYPTO_FAIL | Cryptographic processing failure. |
| FSP_ERR_SB_CRYPTO_AUTH_FAIL | Verification failed. |
| FSP_ERR_SB_CRYPTO_PARAM_ERR | Parameter error. |
| FSP_ERR_SB_CRYPTO_RESOURCE_CONFLICT | CryptoIP is in use. |
| FSP_ERR_UNSUPPORTED | This API is not supported on this device. |

### 5.2.15.8 SCE Protected Mode

Modules » Security

#### Functions

| | |
|---|---|
| fsp_err_t | R_SCE_Open (sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg) |
| fsp_err_t | R_SCE_Close (sce_ctrl_t *const p_ctrl) |
| fsp_err_t | R_SCE_SoftwareReset (void) |
| fsp_err_t | R_SCE_AES128_WrappedKeyGenerate (sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES256_WrappedKeyGenerate (sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA1024_WrappedKeyPairGenerate (sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_RSA2048_WrappedKeyPairGenerate (sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_ECC_secp192r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_ECC_secp224r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| fsp_err_t | R_SCE_RandomNumberGenerate (uint32_t *random) |
| fsp_err_t | R_SCE_AES128_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES256_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128_RFC3394KeyWrap (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_AES256_RFC3394KeyWrap (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key) |
| fsp_err_t | R_SCE_AES128_RFC3394KeyUnwrap (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key) |
| fsp_err_t | R_SCE_AES256_RFC3394KeyUnwrap (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key) |
| fsp_err_t | R_SCE_SHA256HMAC_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA1024_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA1024_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA3072_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSA4096_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128ECB_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128ECB_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES128ECB_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| fsp_err_t | R_SCE_AES128ECB_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128ECB_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES128ECB_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| fsp_err_t | R_SCE_AES256ECB_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES256ECB_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES256ECB_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| fsp_err_t | R_SCE_AES256ECB_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_AES256ECB_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES256ECB_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| fsp_err_t | R_SCE_AES128CBC_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| fsp_err_t | R_SCE_AES128CBC_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES128CBC_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| fsp_err_t | R_SCE_AES128CBC_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| fsp_err_t | R_SCE_AES128CBC_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES128CBC_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| fsp_err_t | R_SCE_AES256CBC_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| fsp_err_t | R_SCE_AES256CBC_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES256CBC_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| fsp_err_t | R_SCE_AES256CBC_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| fsp_err_t | R_SCE_AES256CBC_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES256CBC_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| fsp_err_t | R_SCE_AES128GCM_EncryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| fsp_err_t | R_SCE_AES128GCM_EncryptUpdate (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length) |
| fsp_err_t | R_SCE_AES128GCM_EncryptFinal (sce_gcm_handle_t *handle, uint8_t |

| | |
|---|---|
| | *cipher, uint32_t *cipher_data_length, uint8_t *atag) |
| fsp_err_t | R_SCE_AES128GCM_DecryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| fsp_err_t | R_SCE_AES128GCM_DecryptUpdate (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length) |
| fsp_err_t | R_SCE_AES128GCM_DecryptFinal (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length) |
| fsp_err_t | R_SCE_AES256GCM_EncryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| fsp_err_t | R_SCE_AES256GCM_EncryptUpdate (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length) |
| fsp_err_t | R_SCE_AES256GCM_EncryptFinal (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag) |
| fsp_err_t | R_SCE_AES256GCM_DecryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| fsp_err_t | R_SCE_AES256GCM_DecryptUpdate (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length) |
| fsp_err_t | R_SCE_AES256GCM_DecryptFinal (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length) |
| fsp_err_t | R_SCE_AES128CCM_EncryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES128CCM_EncryptUpdate (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES128CCM_EncryptFinal (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES128CCM_DecryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t |

| | |
|---|---|
| | payload_length, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES128CCM_DecryptUpdate (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES128CCM_DecryptFinal (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES256CCM_EncryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES256CCM_EncryptUpdate (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| fsp_err_t | R_SCE_AES256CCM_EncryptFinal (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES256CCM_DecryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES256CCM_DecryptUpdate (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| fsp_err_t | R_SCE_AES256CCM_DecryptFinal (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES128CMAC_GenerateInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128CMAC_GenerateUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_AES128CMAC_GenerateFinal (sce_cmac_handle_t *handle, uint8_t *mac) |
| fsp_err_t | R_SCE_AES128CMAC_VerifyInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES128CMAC_VerifyUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_AES128CMAC_VerifyFinal (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_AES256CMAC_GenerateInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_AES256CMAC_GenerateUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_AES256CMAC_GenerateFinal (sce_cmac_handle_t *handle, uint8_t *mac) |
| fsp_err_t | R_SCE_AES256CMAC_VerifyInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_AES256CMAC_VerifyUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_AES256CMAC_VerifyFinal (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_SHA256_Init (sce_sha_md5_handle_t *handle) |
| fsp_err_t | R_SCE_SHA256_Update (sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_SHA256_Final (sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length) |
| fsp_err_t | R_SCE_SHA256HMAC_GenerateInit (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_SHA256HMAC_GenerateUpdate (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_SHA256HMAC_GenerateFinal (sce_hmac_sha_handle_t *handle, uint8_t *mac) |
| fsp_err_t | R_SCE_SHA256HMAC_VerifyInit (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_SHA256HMAC_VerifyUpdate (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length) |
| fsp_err_t | R_SCE_SHA256HMAC_VerifyFinal (sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| fsp_err_t | R_SCE_RSASSA_PKCS1024_SignatureGenerate (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| fsp_err_t | R_SCE_RSASSA_PKCS1024_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |

| | |
|---|---|
| fsp_err_t | R_SCE_RSASSA_PKCS2048_SignatureGenerate (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| fsp_err_t | R_SCE_RSASSA_PKCS2048_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| fsp_err_t | R_SCE_RSASSA_PKCS3072_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| fsp_err_t | R_SCE_RSASSA_PKCS4096_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| fsp_err_t | R_SCE_RSAES_PKCS1024_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSAES_PKCS1024_Decrypt (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSAES_PKCS2048_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSAES_PKCS2048_Decrypt (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSAES_PKCS3072_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_RSAES_PKCS4096_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp192r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp224r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_ECDSA_secp256r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp384r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp192r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp224r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp256r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDSA_secp384r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDH_secp256r1_Init (sce_ecdh_handle_t *handle, uint32_t key_type, uint32_t use_key_id) |
| fsp_err_t | R_SCE_ECDH_secp256r1_PublicKeySign (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDH_secp256r1_PublicKeyVerify (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature (sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_ECDH_secp256r1_SharedSecretCalculate (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key) |
| fsp_err_t | R_SCE_ECDH_secp256r1_KeyDerivation (sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall (uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key) |
| fsp_err_t | R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate (sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key) |
| fsp_err_t | R_SCE_TLS_RootCertificateVerify (uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key) |
| fsp_err_t | R_SCE_TLS_CertificateVerify (uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key) |
| fsp_err_t | R_SCE_TLS_PreMasterSecretEncryptWithRSA2048 (uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret) |
| fsp_err_t | R_SCE_TLS_PreMasterSecretGenerateForRSA2048 (uint32_t *sce_pre_master_secret) |
| fsp_err_t | R_SCE_TLS_MasterSecretGenerate (uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret) |
| fsp_err_t | R_SCE_TLS_SessionKeyGenerate (uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector) |
| fsp_err_t | R_SCE_TLS_VerifyDataGenerate (uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data) |
| fsp_err_t | R_SCE_TLS_ServerKeyExchangeVerify (uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1 (uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret) |

### Detailed Description

Driver for the Secure Crypto Engine (SCE9) on RA MCUs.

# Overview

This module provides SCE functions in protected mode.

*Note*

***For a detailed description of the different SCE9 operating modes, refer to Application Note R11AN0498.***

### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| SCE9 (Protected mode) | RA4M2, RA4M3, RA6M4, RA6M5 |

### Features

The SCE module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/256bit
      - CBC 128/256bit
      - GCM 128/256bit
      - CCM 128/256bit
  - Asymmetric Encryption/Decryption
    - RSA
      - RSAES-PKCS1-V1_5 1024/2048bit
      - RSAES-PKCS1-V1_5 3072/4096bit (Encryption only)
      - RSASSA-PKCS1-V1_5 1024/2048bit
      - RSASSA-PKCS1-V1_5 3072/4096bit (Verification only)
    - ECC
      - ECDSA secp192r1/secp224r1/secp256r1/secp384r1
      - ECDH secp256r1
  - Hash Functions
    - SHA-2
      - SHA-256
- Message Authentication Code
  - HMAC-SHA256bit
  - AES-CMAC 128/256bit
- Key Support
  - AES 128/256bit
  - AES Key Wrap/Key Unwrap 128/256bit
  - RSA 1024/2048bit
  - RSA 3072/4096bit (public key only)
  - ECC secp192r1/secp224r1/secp256r1/secp384r1

    - HMAC-SHA256bit
- TRNG
- TLS
    - SSL / TLS support function (TLS1.2 compliant)

# Configuration

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Getting Started: Creating a SCE Protected Mode Project

Start by creating a new project in e² studio or RASC. On the Stacks tab, add New > Security > SCE Protected Mode. For information on how to install and update secure keys, refer to the Application Note R11AN0496.

### Reducing intialization time

The SCE intialization sequence can be modified to support a smaller subset of features and allow for a shorter intialization period. This is particularly useful in cases where startup time is important and the cryptographic features that are required are known in advance. The current fast boot configuration option supports the cryptographic primitives required by MCUBoot including:

- RSA 3K/4K signature verification
- ECC P256 signature verification
- SHA224/256 and GHASH calcuation

### Limitations

### Usage of R_SCE_ECDSA_secp384r1_SignatureGenerate/Verify

The SCE does not support SHA-384 in hardware, so the APIs listed below require the user to create a SHA-384 function for signature generation and verification. To use the APIs listed below, enable SCE_USER_SHA_384_ENABLED on RA Smart Configurator and prepare a function called SCE_USER_SHA_384_FUNCTION. The interface of SCE_USER_SHA_384_FUNCTION, which is called by the following APIs, is described below.

- R_SCE_ECDSA_secp384r1_SignatureGenerate()
- R_SCE_ECDSA_secp384r1_SignatureVerify()

### SCE_USER_SHA_384_FUNCTION()

```
uint32_t SCE_USER_SHA_384_FUNCTION(uint8_t *message, uint8_t* digest, uint32_t
message_length)
```

SHA-384 hash calculation is performed for an area extending the number of bytes specified by the argument message_length from the address specified by the argument message. The calculation result should be stored at the address specified by the argument digest.

Parameters

| | |
|---|---|
| message | [in] Start address of message |
| digest | [in,out] address for storing hash calculation result (48 bytes) |
| message_length | [in] Effective byte count of message |

Return values

| | |
|---|---|
| 0 | Hash value stored successfully. |
| others | Storing of hash value failed. |

# Examples

### AES Example

This is an example of AES-256 encryption and decryption.

```
static uint8_t plain[BLOCK * 2] =
{
    0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
    0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
};
void r_sce_example_aes ()
{
 sce_aes_handle_t      handle;
 sce_aes_wrapped_key_t wrapped_key;
    uint8_t               cipher_calculated[32] = {0};
    uint8_t               plain_calculated[32] = {0};
    uint32_t             dummy;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random key */
 R_SCE_AES256_WrappedKeyGenerate(&wrapped_key);
```

```
 /* Encrypt a plain text */
 R_SCE_AES256ECB_EncryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_EncryptUpdate(&handle, plain, cipher_calculated, BLOCK * 2);
 R_SCE_AES256ECB_EncryptFinal(&handle, cipher_calculated, &dummy);
 /* Decrypt a cipher text using same key as Encryption */
 R_SCE_AES256ECB_DecryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_DecryptUpdate(&handle, cipher_calculated, plain_calculated, BLOCK *
2);
 R_SCE_AES256ECB_DecryptFinal(&handle, plain_calculated, &dummy);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 /* Compare plain and plain_calculated */
 if (memcmp(plain, plain_calculated, BLOCK * 2))
    {
 while (1)
      {
 /* plain and plain_calculated are different (incorrect) */
       }
    }
 else
    {
 while (1)
      {
 /* plain and plain_calculated are the same (correct) */
       }
    }
}
```

### ECC Example

This is an example of ECC secp-256 signature generate and verify.

```
uint8_t ecc256_data_msg[] =
{
 's', 'a', 'm', 'p', 'l', 'e'
```

```
};
void r_sce_example_ecc ()
{
 sce_ecc_wrapped_pair_key_t wrapped_pair_key;
 sce_ecdsa_byte_data_t      message_hash =
    {
       0
    };
 sce_ecdsa_byte_data_t signature;
    uint8_t               out_data[HW_SCE_ECDSA_DATA_BYTE_SIZE];
 fsp_err_t             return_code;
    message_hash.data_length = sizeof(ecc256_data_msg);
    message_hash.pdata       = (uint8_t *) ecc256_data_msg;
    signature.pdata          = out_data;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random pair key */
 R_SCE_ECC_secp256r1_WrappedKeyPairGenerate(&wrapped_pair_key);
 /* Generate a Signature */
 R_SCE_ECDSA_secp256r1_SignatureGenerate(&message_hash, &signature,
&wrapped_pair_key.priv_key);
 /* Verify Signature and Public wrapped key */
    return_code = R_SCE_ECDSA_secp256r1_SignatureVerify(&signature, &message_hash,
&wrapped_pair_key.pub_key);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 if (FSP_SUCCESS != return_code)
    {
 while (1)
     {
 /* Verify Fail */
       }
    }
 else
```

```
    {
 while (1)
      {
 /* Verify Success */
       }
    }
}
```

### RSA Example

This is an example of RSA-2048 encryption and decryption.

```
uint8_t rsa_msg[256] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c,
0x2d, 0x2e, 0x2f,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c,
0x3d, 0x3e, 0x3f,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c,
0x4d, 0x4e, 0x4f,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5a, 0x5b, 0x5c,
0x5d, 0x5e, 0x5f,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c,
0x6d, 0x6e, 0x6f,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7a, 0x7b, 0x7c,
0x7d, 0x7e, 0x7f,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c,
0x8d, 0x8e, 0x8f,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x9c,
0x9d, 0x9e, 0x9f,
    0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xab, 0xac,
```

```
0xad, 0xae, 0xaf,
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xbb, 0xbc,
0xbd, 0xbe, 0xbf,
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc,
0xcd, 0xce, 0xcf,
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xdb, 0xdc,
0xdd, 0xde, 0xdf,
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec,
0xed, 0xee, 0xef,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff
};
void r_sce_example_rsa () {
 sce_rsa2048_wrapped_pair_key_t wrapped_pair_key;
 sce_rsa_byte_data_t            plain;
 sce_rsa_byte_data_t            plain_dec;
 sce_rsa_byte_data_t            cipher;
    uint8_t enc_cipher[HW_SCE_RSA_2048_DATA_BYTE_SIZE];
    uint8_t dec_plain[HW_SCE_RSA_2048_DATA_BYTE_SIZE];
    plain.data_length    = sizeof(rsa_msg) - PADDING_MINIMUM_BYTE_SIZE;
    plain.pdata          = (uint8_t *) rsa_msg;
    plain_dec.data_length = sizeof(dec_plain);
    plain_dec.pdata       = dec_plain;
    cipher.data_length    = sizeof(enc_cipher);
    cipher.pdata          = enc_cipher;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random key */
 R_SCE_RSA2048_WrappedKeyPairGenerate(&wrapped_pair_key);
 /* Encrypt a plain data */
 R_SCE_RSAES_PKCS2048_Encrypt(&plain, &cipher, &wrapped_pair_key.pub_key);
 /* Decrypt a plain data */
 R_SCE_RSAES_PKCS2048_Decrypt(&cipher, &plain_dec, &wrapped_pair_key.priv_key);
 /* SCE power off */
```

```
R_SCE_Close(&sce_ctrl);
/* Compare plain_dec and plain */
if (0 != memcmp(plain_dec.pdata, plain.pdata, plain_dec.data_length))
   {
while (1)
     {
/* plain_dec and plain are different (incorrect) */
      }
   }
else
   {
while (1)
     {
/* plain_dec and plain are the same (correct) */
      }
   }
}
```

### HASH Example

This is an example of calculating the SHA256 hash.

```
uint8_t message[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
};
uint8_t hash[] =
{
    0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
    0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
```

```
};
void r_sce_example_hash ()
{
 sce_sha_md5_handle_t handle;
    uint8_t             digest[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE] = {0};
    uint32_t             digest_length = 0;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Encrypt a message text */
 R_SCE_SHA256_Init(&handle);
 R_SCE_SHA256_Update(&handle, &message, HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE);
 R_SCE_SHA256_Final(&handle, &digest, &digest_length);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 /* Compare digest and hash */
 if (0 != memcmp(digest, hash, digest_length))
    {
 while (1)
     {
 /* digest and hash are different (incorrect) */
      }
    }
 else
    {
 while (1)
     {
 /* digest and hash are the same (correct) */
      }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | sce_instance_ctrl_t |

## Data Structure Documentation

### ◆ sce_instance_ctrl_t

| struct sce_instance_ctrl_t |
| --- |
| SCE private control block. DO NOT MODIFY. Initialization occurs when R_SCE_Open() is called. |

## Function Documentation

### ◆ R_SCE_Open()

fsp_err_t R_SCE_Open ( sce_ctrl_t *const *p_ctrl*, sce_cfg_t const *const *p_cfg* )

Enables use of SCE functionality.

**Parameters**

| | | |
| --- | --- | --- |
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_FAIL | The error-detection self-test failed to terminate normally. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_RETRY | Indicates that an entropy evaluation failure occurred. Run the function again. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

***Note***

*The valid pre-run state is SCE disabled. The pre-run state is SCE Disabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_Close()

fsp_err_t R_SCE_Close ( sce_ctrl_t *const *p_ctrl*)

Stops supply of power to the SCE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |

***Note***

*The pre-run state is any state. After the function runs the state transitions to SCE Disabled State.*

### ◆ R_SCE_SoftwareReset()

fsp_err_t R_SCE_SoftwareReset ( void )

Software reset to SCE.

Reverts the state to the SCE initial state.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |

***Note***

*The pre-run state is any state. After the function runs the state transitions to SCE Disabled State.*

### ◆ R_SCE_AES128_WrappedKeyGenerate()

fsp_err_t R_SCE_AES128_WrappedKeyGenerate ( sce_aes_wrapped_key_t * *wrapped_key*)

This API outputs 128-bit AES wrapped key from a random number.

This API generates a wrapped key from a random number in the SCE. Accordingly, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Disabled State.*

### ◆ R_SCE_AES256_WrappedKeyGenerate()

fsp_err_t R_SCE_AES256_WrappedKeyGenerate ( sce_aes_wrapped_key_t * *wrapped_key*)

This API outputs 256-bit AES wrapped key from a random number.

This API generates a wrapped key from a random number in the SCE. Accordingly, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Disabled State.*

### ◆ R_SCE_RSA1024_WrappedKeyPairGenerate()

fsp_err_t R_SCE_RSA1024_WrappedKeyPairGenerate ( sce_rsa1024_wrapped_pair_key_t * *wrapped_pair_key*)

This API outputs a wrapped key pair for a 1024-bit RSA public key and private key pair. These keys are generated from a random value produced internally by the SCE. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the wrapped key output by this API. A public wrapped key is generated by wrapped_pair_key->pub_key, and a private wrapped key is generated by wrapped_pair_key->priv_key. As the public key exponent, only 0x00010001 is generated.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | User key index for RSA 1024-bit public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. Key generation failed. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_WrappedKeyPairGenerate()

fsp_err_t R_SCE_RSA2048_WrappedKeyPairGenerate ( sce_rsa2048_wrapped_pair_key_t * *wrapped_pair_key*)

This API outputs a wrapped key pair for a 2048-bit RSA public key and private key pair. These keys are generated from a random value produced internally by the SCE. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the wrapped key output by this API. A public wrapped key is generated by wrapped_pair_key->pub_key, and a private wrapped key is generated by wrapped_pair_key->priv_key. As the public key exponent, only 0x00010001 is generated.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | User key index for RSA 2048-bit public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. Key generation failed. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp192r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp192r1_WrappedKeyPairGenerate ( sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp192r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp192r1 public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp224r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp224r1_WrappedKeyPairGenerate ( sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp224r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp224r1 public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp256r1_WrappedKeyPairGenerate ( sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp256r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp256r1 public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp384r1_WrappedKeyPairGenerate ( sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp384r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp384r1 public key and private key pair |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RandomNumberGenerate()

fsp_err_t R_SCE_RandomNumberGenerate ( uint32_t * *random*)

This API can generate 4 words random number.

**Parameters**

| | | |
|---|---|---|
| [in,out] | random | Stores 4words (16 bytes) random data. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128_EncryptedKeyWrap()

fsp_err_t R_SCE_AES128_EncryptedKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_aes_wrapped_key_t * *wrapped_key* )

This API wraps 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256_EncryptedKeyWrap()

fsp_err_t R_SCE_AES256_EncryptedKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_aes_wrapped_key_t * *wrapped_key* )

This API wraps 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128_RFC3394KeyWrap()

fsp_err_t R_SCE_AES128_RFC3394KeyWrap ( sce_aes_wrapped_key_t * *master_key*, uint32_t *target_key_type*, sce_aes_wrapped_key_t * *target_key*, uint32_t * *rfc3394_wrapped_key* )

This API wraps 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-128 key used for wrapping. |
| [in] | target_key_type | Selects key to be wrapped. |
| [in] | target_key | Key to be wrapped. |
| [out] | rfc3394_wrapped_key | Wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Resource conflict. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal user Key Generation Information. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

### ◆ R_SCE_AES256_RFC3394KeyWrap()

fsp_err_t R_SCE_AES256_RFC3394KeyWrap ( sce_aes_wrapped_key_t * *master_key*, uint32_t *target_key_type*, sce_aes_wrapped_key_t * *target_key*, uint32_t * *rfc3394_wrapped_key* )

This API wraps 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-256 key used for wrapping. |
| [in] | target_key_type | Selects key to be wrapped. |
| [in] | target_key | Key to be wrapped. |
| [out] | rfc3394_wrapped_key | Wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Resource conflict. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal user Key Generation Information. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

### ◆ R_SCE_AES128_RFC3394KeyUnwrap()

fsp_err_t R_SCE_AES128_RFC3394KeyUnwrap ( sce_aes_wrapped_key_t * *master_key*, uint32_t *target_key_type*, uint32_t * *rfc3394_wrapped_key*, sce_aes_wrapped_key_t * *target_key* )

This API unwraps 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-128 key used for unwrapping. |
| [in] | target_key_type | Selects key to be unwrapped. |
| [in] | rfc3394_wrapped_key | Wrapped key. |
| [out] | target_key | Key to be unwrapped. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Resource conflict. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal user Key Generation Information. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

### ◆ R_SCE_AES256_RFC3394KeyUnwrap()

fsp_err_t R_SCE_AES256_RFC3394KeyUnwrap ( sce_aes_wrapped_key_t * *master_key*, uint32_t *target_key_type*, uint32_t * *rfc3394_wrapped_key*, sce_aes_wrapped_key_t * *target_key* )

This API unwraps 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-256 key used for unwrapping. |
| [in] | target_key_type | Selects key to be unwrapped. |
| [in] | rfc3394_wrapped_key | Wrapped key. |
| [out] | target_key | Key to be unwrapped. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | Resource conflict. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal user Key Generation Information. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

### ◆ R_SCE_SHA256HMAC_EncryptedKeyWrap()

fsp_err_t R_SCE_SHA256HMAC_EncryptedKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_hmac_sha_wrapped_key_t * *wrapped_key* )

This API wraps HMAC-SHA256 key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | HMAC-SHA256 wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA1024_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA1024_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa1024_public_wrapped_key_t * *wrapped_key* )

This API wraps 1024-bit RSA public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA1024_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_RSA1024_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa1024_private_wrapped_key_t * *wrapped_key* )

This API wraps 1024-bit RSA private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa2048_public_wrapped_key_t * *wrapped_key* )

This API wraps 2048-bit RSA public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa2048_private_wrapped_key_t * *wrapped_key* )

This API wraps 2048-bit RSA private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA3072_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA3072_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa3072_public_wrapped_key_t * *wrapped_key* )

This API wraps 3072-bit RSA public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 3072-bit RSA public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA4096_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA4096_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa4096_public_wrapped_key_t * *wrapped_key* )

This API wraps 4096-bit RSA public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

This API wraps secp192r1 public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp192r1 public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

This API wraps secp224r1 public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp224r1 public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

This API wraps secp256r1 public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp256r1 public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

This API wraps secp384r1 public key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp384r1 public wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

This API wraps secp192r1 private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp192r1 private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

This API wraps secp224r1 private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp224r1 private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

This API wraps secp256r1 private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp256r1 private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap ( uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

This API wraps secp384r1 private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp384r1 private wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_EncryptInit()

fsp_err_t R_SCE_AES128ECB_EncryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES128ECB_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_EncryptUpdate() function and R_SCE_AES128ECB_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_EncryptUpdate()

fsp_err_t R_SCE_AES128ECB_EncryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES128ECB_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES128ECB_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_EncryptFinal()

fsp_err_t R_SCE_AES128ECB_EncryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES128ECB_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_DecryptInit()

fsp_err_t R_SCE_AES128ECB_DecryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES128ECB_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_DecryptUpdate() function and R_SCE_AES128ECB_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_DecryptUpdate()

fsp_err_t R_SCE_AES128ECB_DecryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES128ECB_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES128ECB_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128ECB_DecryptFinal()

fsp_err_t R_SCE_AES128ECB_DecryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES128ECB_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_EncryptInit()

fsp_err_t R_SCE_AES256ECB_EncryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES256ECB_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256ECB_EncryptUpdate() function and R_SCE_AES256ECB_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_EncryptUpdate()

fsp_err_t R_SCE_AES256ECB_EncryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES256ECB_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES256ECB_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_EncryptFinal()

fsp_err_t R_SCE_AES256ECB_EncryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES256ECB_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_DecryptInit()

fsp_err_t R_SCE_AES256ECB_DecryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES128ECB_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_DecryptUpdate() function and R_SCE_AES128ECB_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_DecryptUpdate()

fsp_err_t R_SCE_AES256ECB_DecryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES256ECB_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES256ECB_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256ECB_DecryptFinal()

fsp_err_t R_SCE_AES256ECB_DecryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES256ECB_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_EncryptInit()

fsp_err_t R_SCE_AES128CBC_EncryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector* )

The R_SCE_AES128CBC_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CBC_EncryptUpdate() function and R_SCE_AES128CBC_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_EncryptUpdate()

fsp_err_t R_SCE_AES128CBC_EncryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES128CBC_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES128CBC_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_EncryptFinal()

fsp_err_t R_SCE_AES128CBC_EncryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES128CBC_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_DecryptInit()

fsp_err_t R_SCE_AES128CBC_DecryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector* )

The R_SCE_AES128CBC_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CBC_DecryptUpdate() function and R_SCE_AES128CBC_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_DecryptUpdate()

fsp_err_t R_SCE_AES128CBC_DecryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES128CBC_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES128CBC_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CBC_DecryptFinal()

fsp_err_t R_SCE_AES128CBC_DecryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES128CBC_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_EncryptInit()

fsp_err_t R_SCE_AES256CBC_EncryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector* )

The R_SCE_AES256CBC_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CBC_EncryptUpdate() function and R_SCE_AES256CBC_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initial vector area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_EncryptUpdate()

fsp_err_t R_SCE_AES256CBC_EncryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES256CBC_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES256CBC_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_EncryptFinal()

fsp_err_t R_SCE_AES256CBC_EncryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES256CBC_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_DecryptInit()

fsp_err_t R_SCE_AES256CBC_DecryptInit ( sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector* )

The R_SCE_AES256CBC_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CBC_DecryptUpdate() function and R_SCE_AES256CBC_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Input illegal wrapped key. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_DecryptUpdate()

fsp_err_t R_SCE_AES256CBC_DecryptUpdate ( sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES256CBC_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES256CBC_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CBC_DecryptFinal()

fsp_err_t R_SCE_AES256CBC_DecryptFinal ( sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length* )

Using the handle specified in the first argument, handle, the R_SCE_AES256CBC_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_EncryptInit()

fsp_err_t R_SCE_AES128GCM_EncryptInit ( sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length* )

The R_SCE_AES128GCM_EncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128GCM_EncryptUpdate() function and R_SCE_AES128GCM_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_EncryptUpdate()

fsp_err_t R_SCE_AES128GCM_EncryptUpdate ( sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_data_length*, uint8_t * *aad*, uint32_t *aad_length* )

The R_SCE_AES128GCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES128GCM_EncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state.

Specify areas for plain and cipher that do not overlap. For plain, cipher, initial_vector, and aad, specify RAM addresses that are multiples of 4

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_data_length | plaintext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | After the data from plain was input, an invalid handle was input from aad. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_EncryptFinal()

fsp_err_t R_SCE_AES128GCM_EncryptFinal ( sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_data_length*, uint8_t * *atag* )

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_SCE_AES128GCM_EncryptUpdate (), the R_SCE_AES128GCM_EncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area (cipher_data_length byte) |
| [in,out] | cipher_data_length | ciphertext data length (0 always written here) |
| [in,out] | atag | authentication tag area |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_DecryptInit()

fsp_err_t R_SCE_AES128GCM_DecryptInit ( sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length* )

The R_SCE_AES128GCM_DecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128GCM_DecryptUpdate() function and R_SCE_AES128GCM_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_DecryptUpdate()

fsp_err_t R_SCE_AES128GCM_DecryptUpdate ( sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_data_length*, uint8_t * *aad*, uint32_t *aad_length* )

The R_SCE_AES128GCM_DecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES128GCM_DecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, stage, initial_vector, and aad, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in] | plain | plaintext data area |
| [in] | cipher_data_length | ciphertext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | After the data from plain was input, an invalid handle was input from aad. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128GCM_DecryptFinal()

fsp_err_t R_SCE_AES128GCM_DecryptFinal ( sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_data_length*, uint8_t * *atag*, uint32_t *atag_length* )

The R_SCE_AES128GCM_DecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_SCE_AES128GCM_DecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_length. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | plain | plaintext data area (cipher_data_length byte) |
| [in,out] | plain_data_length | plaintext data length (0 always written here) |
| [in,out] | atag | authentication tag area (atag_length byte) |
| [in] | atag_length | authentication tag length (4,8,12,13,14,15,16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_EncryptInit()

fsp_err_t R_SCE_AES256GCM_EncryptInit ( sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length* )

The R_SCE_AES256GCM_EncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256GCM_EncryptUpdate() function and R_SCE_AES256GCM_EncryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_EncryptUpdate()

fsp_err_t R_SCE_AES256GCM_EncryptUpdate ( sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_data_length*, uint8_t * *aad*, uint32_t *aad_length* )

The R_SCE_AES256GCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES256GCM_EncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state.

Specify areas for plain and cipher that do not overlap. For plain, cipher, initial_vector, and aad, specify RAM addresses that are multiples of 4

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_data_length | plaintext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | After the data from plain was input, an invalid handle was input from aad. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_EncryptFinal()

fsp_err_t R_SCE_AES256GCM_EncryptFinal ( sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_data_length*, uint8_t * *atag* )

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_SCE_AES256GCM_EncryptUpdate (), the R_SCE_AES256GCM_EncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area (cipher_data_length byte) |
| [in,out] | cipher_data_length | ciphertext data length (0 always written here) |
| [in,out] | atag | authentication tag area |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_DecryptInit()

fsp_err_t R_SCE_AES256GCM_DecryptInit ( sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length* )

The R_SCE_AES256GCM_DecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256GCM_DecryptUpdate() function and R_SCE_AES256GCM_DecryptFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_DecryptUpdate()

fsp_err_t R_SCE_AES256GCM_DecryptUpdate ( sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_data_length*, uint8_t * *aad*, uint32_t *aad_length* )

The R_SCE_AES256GCM_DecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES256GCM_DecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, stage, initial_vector, and aad, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in] | plain | plaintext data area |
| [in] | cipher_data_length | ciphertext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | After the data from plain was input, an invalid handle was input from aad. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256GCM_DecryptFinal()

fsp_err_t R_SCE_AES256GCM_DecryptFinal ( sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_data_length*, uint8_t * *atag*, uint32_t *atag_length* )

The R_SCE_AES256GCM_DecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_SCE_AES256GCM_DecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_length. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | plain | plaintext data area (cipher_data_length byte) |
| [in,out] | plain_data_length | plaintext data length (0 always written here) |
| [in,out] | atag | authentication tag area (atag_length byte) |
| [in] | atag_length | authentication tag length (4,8,12,13,14,15,16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_EncryptInit()

fsp_err_t R_SCE_AES128CCM_EncryptInit ( sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length* )

The R_SCE_AES128CCM_EncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES128CCM_EncryptUpdate() and R_SCE_AES128CCM_EncryptFinal() use handle as an argument.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_EncryptUpdate()

fsp_err_t R_SCE_AES128CCM_EncryptUpdate ( sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES128CCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in CCM mode using the values specified by wrapped_key, nonce, and adata in R_SCE_AES128CCM_EncryptInit(). This function buffers internally the data input by the user until the input value of plain exceeds 16 bytes. Once the amount of plain input data is 16 bytes or greater, the encrypted result is output to cipher, which is specified in the third argument. Use payload_length in R_SCE_AES128CCM_EncryptInit() to specify the total data length of plain that will be input. Use plain_length in this function to specify the data length to be input when the user calls this function. If the input value of plain is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_length | plaintext data length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_EncryptFinal()

fsp_err_t R_SCE_AES128CCM_EncryptFinal ( sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*, uint8_t * *mac*, uint32_t *mac_length* )

If the data length of plain input in R_SCE_AES128CCM_EncryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES128CCM_EncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length to the same value as that specified for the argument mac_length in Aes128CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in,out] | cipher_length | ciphertext data length |
| [in,out] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_DecryptInit()

fsp_err_t R_SCE_AES128CCM_DecryptInit ( sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length* )

The R_SCE_AES128CCM_DecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES128CCM_DecryptUpdate() and R_SCE_AES128CCM_DecryptFinal() use handle as an argument.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_DecryptUpdate()

fsp_err_t R_SCE_AES128CCM_DecryptUpdate ( sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES128CCM_DecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by wrapped_key, nonce, and adata in in R_SCE_AES128CCM_DecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_length in R_SCE_AES128CCM_DecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in] | cipher_length | ciphertext data length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CCM_DecryptFinal()

fsp_err_t R_SCE_AES128CCM_DecryptFinal ( sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*, uint8_t * *mac*, uint32_t *mac_length* )

If the data length of cipher input in R_SCE_AES128GCM_DecryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES128GCM_DecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_length in Aes128CcmDecryptInit().

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | plain | plaintext data area |
| [in,out] | plain_length | plaintext data length |
| [in] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error, or authentication failed. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_EncryptInit()

fsp_err_t R_SCE_AES256CCM_EncryptInit ( sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length* )

The R_SCE_AES256CCM_EncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES256CCM_EncryptUpdate() and R_SCE_AES256CCM_EncryptFinal() use handle as an argument.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_EncryptUpdate()

fsp_err_t R_SCE_AES256CCM_EncryptUpdate ( sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length* )

The R_SCE_AES256CCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in CCM mode using the values specified by wrapped_key, nonce, and adata in R_SCE_AES256CCM_EncryptInit(). This function buffers internally the data input by the user until the input value of plain exceeds 16 bytes. Once the amount of plain input data is 16 bytes or greater, the encrypted result is output to cipher, which is specified in the third argument. Use payload_length in R_SCE_AES256CCM_EncryptInit() to specify the total data length of plain that will be input. Use plain_length in this function to specify the data length to be input when the user calls this function. If the input value of plain is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_length | plaintext data length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_EncryptFinal()

fsp_err_t R_SCE_AES256CCM_EncryptFinal ( sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*, uint8_t * *mac*, uint32_t *mac_length* )

If the data length of plain input in R_SCE_AES256CCM_EncryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES256CCM_EncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length to the same value as that specified for the argument mac_length in Aes256CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in,out] | cipher_length | ciphertext data length |
| [in,out] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_DecryptInit()

fsp_err_t R_SCE_AES256CCM_DecryptInit ( sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length* )

The R_SCE_AES256CCM_DecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES256CCM_DecryptUpdate() and R_SCE_AES256CCM_DecryptFinal() use handle as an argument.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_DecryptUpdate()

fsp_err_t R_SCE_AES256CCM_DecryptUpdate ( sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length* )

The R_SCE_AES256CCM_DecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by wrapped_key, nonce, and adata in in R_SCE_AES256CCM_DecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_length in R_SCE_AES256CCM_DecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in] | cipher_length | ciphertext data length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CCM_DecryptFinal()

fsp_err_t R_SCE_AES256CCM_DecryptFinal ( sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*, uint8_t * *mac*, uint32_t *mac_length* )

If the data length of cipher input in R_SCE_AES256GCM_DecryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES256GCM_DecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_length in Aes256CcmDecryptInit().

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | plain | plaintext data area |
| [in,out] | plain_length | plaintext data length |
| [in] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error, or authentication failed. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_GenerateInit()

fsp_err_t R_SCE_AES128CMAC_GenerateInit ( sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES128CMAC_GenerateInit() function performs preparations for the execution of an CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CMAC_GenerateUpdate() function and R_SCE_AES128CMAC_GenerateFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_GenerateUpdate()

fsp_err_t R_SCE_AES128CMAC_GenerateUpdate ( sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_AES128CMAC_GenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES128CMAC_GenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_GenerateFinal()

fsp_err_t R_SCE_AES128CMAC_GenerateFinal ( sce_cmac_handle_t * *handle*, uint8_t * *mac* )

The R_SCE_AES128CMAC_GenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Not used. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_VerifyInit()

fsp_err_t R_SCE_AES128CMAC_VerifyInit ( sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES128CMAC_VerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argumentin the subsequent R_SCE_AES128CMAC_VerifyUpdate() function and R_SCE_AES128CMAC_VerifyFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_VerifyUpdate()

fsp_err_t R_SCE_AES128CMAC_VerifyUpdate ( sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_AES128CMAC_VerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES128CMAC_VerifyInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128CMAC_VerifyFinal()

fsp_err_t R_SCE_AES128CMAC_VerifyFinal ( sce_cmac_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length* )

The R_SCE_AES128CMAC_VerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (mac_length byte) |
| [in,out] | mac_length | MAC data length (2 to 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_GenerateInit()

fsp_err_t R_SCE_AES256CMAC_GenerateInit ( sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES256CMAC_GenerateInit() function performs preparations for the execution of an CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CMAC_GenerateUpdate() function and R_SCE_AES256CMAC_GenerateFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_GenerateUpdate()

fsp_err_t R_SCE_AES256CMAC_GenerateUpdate ( sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_AES256CMAC_GenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES256CMAC_GenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_GenerateFinal()

fsp_err_t R_SCE_AES256CMAC_GenerateFinal ( sce_cmac_handle_t * *handle*, uint8_t * *mac* )

The R_SCE_AES256CMAC_GenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (16byte) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Not used. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_VerifyInit()

fsp_err_t R_SCE_AES256CMAC_VerifyInit ( sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_AES256CMAC_VerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argumentin the subsequent R_SCE_AES256CMAC_VerifyUpdate() function and R_SCE_AES256CMAC_VerifyFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_VerifyUpdate()

fsp_err_t R_SCE_AES256CMAC_VerifyUpdate ( sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_AES256CMAC_VerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES256CMAC_VerifyInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256CMAC_VerifyFinal()

fsp_err_t R_SCE_AES256CMAC_VerifyFinal ( sce_cmac_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length* )

The R_SCE_AES256CMAC_VerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (mac_length byte) |
| [in,out] | mac_length | MAC data length (2 to 16 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_ASSERTION | A required parameter is NULL. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256_Init()

fsp_err_t R_SCE_SHA256_Init ( sce_sha_md5_handle_t * *handle*)

The R_SCE_SHA256_Init() function performs preparations for the execution of an SHA-256 hash calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_SHA256_Update() function and R_SCE_SHA256_Final() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256_Update()

fsp_err_t R_SCE_SHA256_Update ( sce_sha_md5_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_SHA256_Update() function calculates a hash value based on the second argument, message, and the third argument, message_length, and writes the ongoing status to the first argument, handle. After message input is completed, call R_SCE_SHA256_Final().

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |
| [in] | message | message data area |
| [in] | message_length | message data length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256_Final()

fsp_err_t R_SCE_SHA256_Final ( sce_sha_md5_handle_t * *handle*, uint8_t * *digest*, uint32_t * *digest_length* )

Using the handle specified in the first argument, handle, the R_SCE_SHA256_Final() function writes the calculation result to the second argument, digest, and writes the length of the calculation result to the third argument, digest_length.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |
| [in,out] | digest | hasha data area |
| [in,out] | digest_length | hash data length (32bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_GenerateInit()

fsp_err_t R_SCE_SHA256HMAC_GenerateInit ( sce_hmac_sha_handle_t * *handle*, sce_hmac_sha_wrapped_key_t * *wrapped_key* )

The R_SCE_SHA256HMAC_GenerateInit() function uses the second argument wrapped_key to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. The argument handle is used by the subsequent R_SCE_SHA256HMAC_GenerateUpdate() function or R_SCE_SHA256HMAC_GenerateFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handler (work area) |
| [in] | wrapped_key | MAC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | An invalid MAC wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_GenerateUpdate()

fsp_err_t R_SCE_SHA256HMAC_GenerateUpdate ( sce_hmac_sha_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_SHA256HMAC_GenerateUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_SCE_SHA256HMAC_GenerateFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | message | Message area |
| [in] | message_length | Message length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_GenerateFinal()

fsp_err_t R_SCE_SHA256HMAC_GenerateFinal ( sce_hmac_sha_handle_t * *handle*, uint8_t * *mac* )

The R_SCE_SHA256HMAC_GenerateFinal() function uses the handle specified by the first argument handle and writes the calculation result to the second argument mac.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in,out] | mac | HMAC area (32 bytes) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_VerifyInit()

fsp_err_t R_SCE_SHA256HMAC_VerifyInit ( sce_hmac_sha_handle_t * *handle*, sce_hmac_sha_wrapped_key_t * *wrapped_key* )

The R_SCE_SHA256HMAC_VerifyInit() function uses the second argument wrapped_key to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. The argument handle is used by the subsequent R_SCE_SHA256HMAC_VerifyUpdate() function or R_SCE_SHA256HMAC_VerifyFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handler (work area) |
| [in] | wrapped_key | MAC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | An invalid MAC wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_VerifyUpdate()

fsp_err_t R_SCE_SHA256HMAC_VerifyUpdate ( sce_hmac_sha_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length* )

The R_SCE_SHA256HMAC_VerifyUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_SCE_SHA256HMAC_VerifyFinal() function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | message | Message area |
| [in] | message_length | Message length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_SHA256HMAC_VerifyFinal()

fsp_err_t R_SCE_SHA256HMAC_VerifyFinal ( sce_hmac_sha_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length* )

The R_SCE_SHA256HMAC_VerifyFinal() function uses the handle specified by the first argument handle and verifies the mac value from the second argument mac and third argument mac_length. Input a value in bytes from 4 to 32 as mac_length.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | mac | HMAC area |
| [in] | mac_length | HMAC length |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS1024_SignatureGenerate()

fsp_err_t R_SCE_RSASSA_PKCS1024_SignatureGenerate ( sce_rsa_byte_data_t * *message_hash*, sce_rsa_byte_data_t * *signature*, sce_rsa1024_private_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS1024_SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private wrapped key input to the third argument, wrapped_key, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value |

| | | |
|---|---|---|
| | | • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : data length |
| [in] | wrapped_key | Inputs the 1024-bit RSA private wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS1024_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS1024_SignatureVerify ( sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa1024_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS1024_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : Specifies effective data length of the array |
| [in] | message_hash | Message text or hash value to verify<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |
| [in] | hash_type | Only |

| | | |
|---|---|---|
| | | HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS2048_SignatureGenerate()

fsp_err_t R_SCE_RSASSA_PKCS2048_SignatureGenerate ( sce_rsa_byte_data_t * *message_hash*, sce_rsa_byte_data_t * *signature*, sce_rsa2048_private_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS2048_SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private wrapped key input to the third argument, wrapped_key, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the |

| | | |
|---|---|---|
| | | data type of message_hash (Message: 0 Hash value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : data length |
| [in] | wrapped_key | Inputs the 2048-bit RSA private wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS2048_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS2048_SignatureVerify ( sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa2048_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS2048_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : Specifies effective data length of the array |
| [in] | message_hash | Message text or hash value to verify<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |
| [in] | hash_type | Only |

| | | |
|---|---|---|
| | | HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS3072_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS3072_SignatureVerify ( sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa3072_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS3072_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : Specifies effective data length of the array |
| [in] | message_hash | Message text or hash value to verify<br>• message_hash->pda |

| | | |
|---|---|---|
| | | ta : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Inputs the 3072-bit RSA public wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSASSA_PKCS4096_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS4096_SignatureVerify ( sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa4096_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type* )

The R_SCE_RSASSA_PKCS4096_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify<br>• signature->pdata : Specifies pointer to array storing the signature text<br>• signature->data_length : Specifies effective data length of the array |
| [in] | message_hash | Message text or hash value to verify<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |
| [in] | hash_type | Only |

| | | |
|---|---|---|
| | | HW_SCE_RSA_HASH_SHA256 is supported |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_AUTHENTICATION | Authentication failed |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS1024_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS1024_Encrypt ( sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa1024_public_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS1024_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11 |
| [in,out] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS1024_Decrypt()

fsp_err_t R_SCE_RSAES_PKCS1024_Decrypt ( sce_rsa_byte_data_t * *cipher*, sce_rsa_byte_data_t * *plain*, sce_rsa1024_private_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS1024_Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

**Parameters**

| | | |
|---|---|---|
| [in] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in,out] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11. Outputs valid data length after decryption (public key n size). |
| [in] | wrapped_key | Inputs the 1024-bit RSA private wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |

| | |
|---|---|
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS2048_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS2048_Encrypt ( sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa2048_public_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS2048_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11 |
| [in,out] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in] | wrapped_key | Inputs the 2048-bit RSA public wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS2048_Decrypt()

fsp_err_t R_SCE_RSAES_PKCS2048_Decrypt ( sce_rsa_byte_data_t * *cipher*, sce_rsa_byte_data_t * *plain*, sce_rsa2048_private_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS2048_Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

**Parameters**

| | | |
|---|---|---|
| [in] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in,out] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11. Outputs valid data length after decryption (public key n size). |
| [in] | wrapped_key | Inputs the 1024-bit RSA private wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |

| | |
|---|---|
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS3072_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS3072_Encrypt ( sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa3072_public_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS3072_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11 |
| [in,out] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in] | wrapped_key | Inputs the 3072-bit RSA public wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSAES_PKCS4096_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS4096_Encrypt ( sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa4096_public_wrapped_key_t * *wrapped_key* )

The R_SCE_RSAES_PKCS4096_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext<br>• plain->pdata : Specifies pointer to array containing plaintext.<br>• plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11 |
| [in,out] | cipher | ciphertext<br>• cipher->pdata : Specifies pointer to array containing ciphertext.<br>• cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size). |
| [in] | wrapped_key | Inputs the 4096-bit RSA public wrapped key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Incorrect wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is illegal. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp192r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp192r1_SignatureGenerate ( sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp192r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp192r1 using the private wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) \|\| signature r (192 bits) \|\| 0 padding (64 bits) \|\| signature s (192 bits)".<br>• signature->data_length : Data length (byte units) |
| | | |

| | | |
|---|---|---|
| [in] | wrapped_key | Input wrapped key of secp192r1 private key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp224r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp224r1_SignatureGenerate ( sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp224r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp224r1 using the private wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash |

| | | |
|---|---|---|
| | | (Message: 0 Hash value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) \|\| signature r (224 bits) \|\| 0 padding (32 bits) \|\| signature s (224 bits)".<br>• signature->data_length : Data length (byte units) |
| [in] | wrapped_key | Input wrapped key of secp224r1 private key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp256r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp256r1_SignatureGenerate ( sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp256r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 32 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp256r1 using the private wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (256 bits) \|\| signature s (256 bits)".<br>• signature->data_length : Data length (byte units) |
| [in] | wrapped_key | Input wrapped key of secp256r1 private key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp384r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp384r1_SignatureGenerate ( sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

When a message is specified in the first argument, message_hash->data_type, a SHA-384 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp384r1 using the private wrapped key input as the third argument, wrapped_key.

To use message input, prepare a user-defined function for SHA384.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 48 bytes of the SHA-384 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp384r1 using the private wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash |

| | | |
|---|---|---|
| | | value: 1) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (384 bits) \|\| signature s (384 bits)".<br>• signature->data_length : Data length (byte units) |
| [in] | wrapped_key | Input wrapped key of secp384r1 private key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp192r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp192r1_SignatureVerify ( sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp192r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp192r1 using the public wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) \|\| signature r (192 bits) \|\| 0 padding (64 bits) \|\| signature s (192 bits)".<br>• signature->data_length : Specifies the data length (byte units) (nonuse) |
| [in,out] | message_hash | Message or hash value to be verified<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |

| | | |
|---|---|---|
| [in] | wrapped_key | Input wrapped key of secp192r1 public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred or signature verification failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp224r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp224r1_SignatureVerify ( sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp224r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp224r1 using the public wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) \|\| signature r (224 bits) \|\| 0 padding (32 bits) \|\| signature s (224 bits)".<br>• signature->data_length : Specifies the data length (byte units) (nonuse) |

| | | |
|---|---|---|
| [in,out] | message_hash | Message or hash value to be verified<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Input wrapped key of secp224r1 public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred or signature verification failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp256r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp256r1_SignatureVerify ( sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp256r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 32 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp256r1 using the public wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (256 bits) \|\| signature s (256 bits)".<br>• signature->data_length : Specifies the data length (byte units) (nonuse) |
| [in,out] | message_hash | Message or hash value to be verified<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Input wrapped key of |

| | | |
|---|---|---|
| | | secp256r1 public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred or signature verification failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDSA_secp384r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp384r1_SignatureVerify ( sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp384r1 using the public wrapped key input as the third argument, wrapped_key.

To use message input, prepare a user-defined function for SHA384.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 48 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp384r1 using the public wrapped key input as the third argument, wrapped_key.

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified<br>• signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (384 bits) \|\| signature s (384 bits)".<br>• signature->data_length : Specifies the data length (byte units) (nonuse) |
| | | |

| | | |
|---|---|---|
| [in,out] | message_hash | Message or hash value to be verified<br>• message_hash->pdata : Specifies pointer to array storing the message or hash value<br>• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected)<br>• message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1) |
| [in] | wrapped_key | Input wrapped key of secp384r1 public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred or signature verification failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDH_secp256r1_Init()

fsp_err_t R_SCE_ECDH_secp256r1_Init ( sce_ecdh_handle_t * *handle*, uint32_t *key_type*, uint32_t *use_key_id* )

The R_SCE_ECDH_secp256r1_Init() function prepares to perform ECDH key exchange computation and writes the result to the first argument, handle. The succeeding functions R_SCE_ECDH_secp256r1_PublicKeySign(), R_SCE_ECDH_secp256r1_PublicKeyVerify(), R_SCE_ECDH_secp256r1_SharedSecretCalculate(), and R_SCE_ECDH_secp256r1_KeyDerivation() use handle as an argument.

Use the second argument, key_type, to select the type of ECDH key exchange. When ECDHE is selected, the R_SCE_ECDH_secp256r1_PublicKeySign() function uses the SCE's random number generation functionality to generate an secp256r1 key pair. When ECDH is selected, keys installed beforehand are used for key exchange.

Input 1 as the third argument, use_key_id, to use key_id when key exchange is performed. key_id is for applications conforming to the DLMS/COSEM standard for smart meters.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | key_type | Key exchange type (0: ECDHE, 1: ECDH, 2:ECDH(AES-GCM-128 with IV)) |
| [in] | use_key_id | 0: key_id not used, 1: key_id used |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_PARAMETER | Input data is invalid. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDH_secp256r1_PublicKeySign()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeySign ( sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, sce_ecc_private_wrapped_key_t * *ecc_private_wrapped_key*, uint8_t * *public_key*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key* )

The R_SCE_ECDH_secp256r1_PublicKeySign() function calculates a signature for a public key user wrapped key used for ECDH key exchange.

If ECDHE is specified by the key_type argument of the R_SCE_ECDH_secp256r1_Init() function, the SCE's random number generation functionality is used to generate an secp256r1 key pair. The public key is output to public_key and the private key is output to wrapped_key.

If ECDH is specified by the key_type argument of the R_SCE_ECDH_secp256r1_Init() function, the public key input as ecc_public_wrapped_key is output to public_key and nothing is output to wrapped_key.

The succeeding function R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses the first argument, handle, as an argument. R_SCE_ECDH_secp256r1_SharedSecretCalculate() function uses wrapped_key as input to calculate Z.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) When using key_id, input handle->key_id after running R_SCE_ECDH_secp256r1_Init(). |
| [in] | ecc_public_wrapped_key | For ECDHE, input a null pointer. For ECDH, input the wrapped key of a secp256r1 public key. |
| [in] | ecc_private_wrapped_key | secp256r1 private key for signature generation |
| [in,out] | public_key | User secp256r1 public key (512-bit) for key exchange. When using key_id, key_id (8-bit) \|\| public key (512-bit) \|\| 0 padding (24-bit) |
| [in,out] | signature | Signature text storage destination information<br>• signature->pdata : Specifies pointer to array storing signature text. The signature format is "signature r (256 bits) \|\| signature s (256 bits)"<br>• signature->data_leng |

| | | |
|---|---|---|
| | | th : Data length (in byte units) |
| [in,out] | wrapped_key | For ECDHE, a private wrapped key generated from a random number. Not output for ECDH. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDH_secp256r1_PublicKeyVerify()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeyVerify ( sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, uint8_t * *public_key_data*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

The R_SCE_ECDH_secp256r1_PublicKeyVerify() function verifies the signature of the secp256r1 public key of the other ECDH key exchange party. If the signature is correct, it outputs the public wrapped key to the fifth argument. The first argument, handle, is used as an argument in the subsequent function R_SCE_ECDH_secp256r1_SharedSecretCalculate(). R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses wrapped_key as input to calculate Z.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | ecc_public_wrapped_key | Public wrapped key area for signature verification |
| [in] | public_key_data | secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) \|\| public key (512-bit) |
| [in] | signature | ECDSA secp256r1 signature of ecc_public_wrapped_key |
| [in,out] | wrapped_key | wrapped key of ecc_public_wrapped_key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred or signature verification failed. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature ( sce_ecdh_handle_t * *handle*, uint8_t * *public_key_data*, sce_ecc_public_wrapped_key_t * *wrapped_key* )

The R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature() function reads the secp256r1 public key of the other ECDH key exchange party and outputs the public wrapped key to the third argument. The first argument, handle, is used as an argument in the subsequent function R_SCE_ECDH_secp256r1_SharedSecretCalculate().
R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses wrapped_key as input to calculate Z. This API does not verify signature of public_key_data, please protect this data by upper layer software.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | public_key_data | secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) \|\| public key (512-bit) |
| [in,out] | wrapped_key | wrapped key of public_key_data |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State. Please note that this is slightly contrary to the protected mode policy as it omits signature verification.*

### ◆ R_SCE_ECDH_secp256r1_SharedSecretCalculate()

fsp_err_t R_SCE_ECDH_secp256r1_SharedSecretCalculate ( sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, sce_ecc_private_wrapped_key_t * *ecc_private_wrapped_key*, sce_ecdh_wrapped_key_t * *shared_secret_wrapped_key* )

The R_SCE_ECDH_secp256r1_SharedSecretCalculate() function uses the ECDH key exchange algorithm to output the wrapped key of the shared secret Z derived from the public key of the other key exchange party and your own private key. Input as the second argument, ecc_public_wrapped_key, the public wrapped key whose signature was verified by R_SCE_ECDH_secp256r1_PublicKeyVerify(). When key_type of R_SCE_ECDH_secp256r1_Init() is 0, input as the third argument, ecc_private_wrapped_key, the private wrapped key generated from a random number by R_SCE_ECDH_secp256r1_PublicKeySign(), and when key_type is other than 0, input the private wrapped key that forms a pair with the second argument of R_SCE_ECDH_secp256r1_PublicKeySign(). The subsequent R_SCE_ECDH_secp256r1_KeyDerivation() function uses shared_secret_wrapped_key as key material for outputting the wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | ecc_public_wrapped_key | Public wrapped key whose signature was verified by R_SCE_ECDH_secp256r1_PublicKeyVerify() |
| [in] | ecc_private_wrapped_key | Private wrapped key |
| [in,out] | shared_secret_wrapped_key | Wrapped key of shared secret Z calculated by ECDH key exchange |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECDH_secp256r1_KeyDerivation()

fsp_err_t R_SCE_ECDH_secp256r1_KeyDerivation ( sce_ecdh_handle_t * *handle*, sce_ecdh_wrapped_key_t * *shared_secret_wrapped_key*, uint32_t *key_type*, uint32_t *kdf_type*, uint8_t * *other_info*, uint32_t *other_info_length*, sce_hmac_sha_wrapped_key_t * *salt_wrapped_key*, sce_aes_wrapped_key_t * *wrapped_key* )

The R_SCE_ECDH_secp256r1_KeyDerivation() function uses the shared secret "Z (shared_secret_index)" calculated by the R_SCE_ECDH_secp256r1_SharedSecretCalculate() function as the key material to derive the wrapped key specified by the third argument, key_type. The key derivation algorithm is one-step key derivation as defined in NIST SP800-56C. Either SHA-256 or SHA-256 HMAC is specified by the fourth argument, kdf_type. When SHA-256 HMAC is specified, the wrapped key output by the R_SCE_SHA256HMAC_EncryptedKeyWrap() function is specified as the seventh argument, salt_wrapped_key. Enter a fixed value for deriving a key shared with the key exchange partner in the fifth argument, other_info. A wrapped key corresponding to key_type is output as the eighth argument, wrapped_key. The correspondences between the types of derived wrapped_key and the functions with which they can be used as listed below.

- AES-128: All AES-128 Init functions
- AES-256: All AES-256 Init functions
- SHA256-HMAC: R_SCE_SHA256HMAC_GenerateInit() function and R_SCE_SHA256HMAC_VerifyInit() function

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | shared_secret_wrapped_key | Z wrapped key calculated by R_SCE_ECDH_secp256r1_SharedSecretCalculate |
| [in] | key_type | Derived key type (0: AES-128, 1: AES-256, 2:SHA256-HMAC, 3: AES-GCM-128 with IV) |
| [in] | kdf_type | Algorithm used for key derivation calculation (0: SHA-256, 1:SHA256-HMAC) |
| [in] | other_info | Additional data used for key derivation calculation: AlgorithmID \|\| PartyUInfo \|\| PartyVInfo |
| [in] | other_info_length | Data length of other_info (up to 147 byte units) |
| [in] | salt_wrapped_key | Salt wrapped key (Input NULL when kdf_type is 0.) |
| [in,out] | wrapped_key | Wrapped key corresponding to key_type. When the value of key_type is 2, an SHA256-HMAC wrapped key is output. wrapped_key can be specified by casting the start address of the area |

| | | |
|---|---|---|
| | | reserved beforehand by the sce_hmac_sha_wrapped_key_t type with the (sce_aes_wrapped_key_t*) type. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal end |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource required by the processing is in use by other processing. |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Invalid wrapped key was input. |
| FSP_ERR_CRYPTO_SCE_PARAMETER | An invalid handle was input. |
| FSP_ERR_CRYPTO_SCE_FAIL | Internal error occurred. |
| FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION | An invalid function was called. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall()

fsp_err_t R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall ( uint8_t * *encrypted_provisioning_key*, uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_tls_ca_certification_public_wrapped_key_t * *wrapped_key* )

Generate TLS RSA Public key index data

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_provisioning_key | the provisioning key includes encrypted CBC/CBC-MAC key for user key |
| [in] | initial_vector | the initial_vector for user key CBC encrypt |
| [in] | encrypted_key | the user key encrypted with AES128-ECB mode |
| [out] | wrapped_key | the user Key Generation Information (141 words) of RSA2048 bit |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate()

fsp_err_t R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate ( sce_tls_p256_ecc_wrapped_key_t * *tls_p256_ecc_wrapped_key*, uint8_t * *ephemeral_ecdh_public_key* )

Generate TLS ECC key pair

**Parameters**

| | | |
|---|---|---|
| [in] | tls_p256_ecc_wrapped_key | P256 ECC key index for TLS |
| [in] | ephemeral_ecdh_public_key | ephemeral ECDH public key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_RootCertificateVerify()

fsp_err_t R_SCE_TLS_RootCertificateVerify ( uint32_t *public_key_type*, uint8_t * *certificate*, uint32_t *certificate_length*, uint32_t *public_key_n_start_position*, uint32_t *public_key_n_end_position*, uint32_t *public_key_e_start_position*, uint32_t *public_key_e_end_position*, uint8_t * *signature*, uint32_t * *encrypted_root_public_key* )

Verify root CA certificate.

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | certificate | certificates. |
| [in] | certificate_length | byte size of certificates. |
| [in] | public_key_n_start_position | start position of public key n. |
| [in] | public_key_n_end_position | end position of public key n. |
| [in] | public_key_e_start_position | start position of public key e. |
| [in] | public_key_e_end_position | end position of public key e. |
| [in] | signature | signature for certificates. |
| [out] | encrypted_root_public_key | public key for RSA 2048bit. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_CertificateVerify()

fsp_err_t R_SCE_TLS_CertificateVerify ( uint32_t *public_key_type*, uint32_t * *encrypted_input_public_key*, uint8_t * *certificate*, uint32_t *certificate_length*, uint8_t * *signature*, uint32_t *public_key_n_start_position*, uint32_t *public_key_n_end_position*, uint32_t *public_key_e_start_position*, uint32_t *public_key_e_end_position*, uint32_t * *encrypted_output_public_key* )

Verify server certificate and intermediate certificate.

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | encrypted_input_public_key | public key. |
| [in] | certificate | certificates. |
| [in] | certificate_length | byte size of certificates. |
| [in] | signature | signature for certificates. |
| [in] | public_key_n_start_position | start position of public key n. |
| [in] | public_key_n_end_position | end position of public key n. |
| [in] | public_key_e_start_position | start position of public key e. |
| [in] | public_key_e_end_position | end position of public key e. |
| [out] | encrypted_output_public_key | public key for RSA 2048bit. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_PreMasterSecretEncryptWithRSA2048()

fsp_err_t R_SCE_TLS_PreMasterSecretEncryptWithRSA2048 ( uint32_t * *encrypted_public_key*, uint32_t * *sce_pre_master_secret*, uint8_t * *encrypted_pre_master_secret* )

Output the result encrypted pre-master secret with RSA 2048bit

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_public_key | public key data. |
| [in] | sce_pre_master_secret | pre-master secret value. |
| [out] | encrypted_pre_master_secret | the value encrypted pre-master secret. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_PreMasterSecretGenerateForRSA2048()

fsp_err_t R_SCE_TLS_PreMasterSecretGenerateForRSA2048 ( uint32_t * *sce_pre_master_secret*)

Generate encrypted pre-master secret.

**Parameters**

| | | |
|---|---|---|
| [out] | sce_pre_master_secret | pre-master secret value for SCE. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_MasterSecretGenerate()

fsp_err_t R_SCE_TLS_MasterSecretGenerate ( uint32_t *select_cipher_suite*, uint32_t * *sce_pre_master_secret*, uint8_t * *client_random*, uint8_t * *server_random*, uint32_t * *sce_master_secret* )

Generate encrypted master secret.

**Parameters**

| | | |
|---|---|---|
| [in] | select_cipher_suite | cipher suite type |
| [in] | sce_pre_master_secret | pre-master secret value for SCE. |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [out] | sce_master_secret | master secret value with SCE-specific conversion. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_SessionKeyGenerate()

fsp_err_t R_SCE_TLS_SessionKeyGenerate ( uint32_t *select_cipher_suite*, uint32_t * *sce_master_secret*, uint8_t * *client_random*, uint8_t * *server_random*, uint8_t * *nonce_explicit*, sce_hmac_sha_wrapped_key_t * *client_mac_wrapped_key*, sce_hmac_sha_wrapped_key_t * *server_mac_wrapped_key*, sce_aes_wrapped_key_t * *client_crypto_wrapped_key*, sce_aes_wrapped_key_t * *server_crypto_wrapped_key*, uint8_t * *client_initial_vector*, uint8_t * *server_initial_vector* )

Output various key information.

**Parameters**

| | | |
|---|---|---|
| [in] | select_cipher_suite | Key suite information number. |
| [in] | sce_master_secret | master secret value. |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [in] | nonce_explicit | nonce value |
| [out] | client_mac_wrapped_key | the mac key during communication from client to server. |
| [out] | server_mac_wrapped_key | the mac key during communication from server to client. |
| [out] | client_crypto_wrapped_key | the crypto key during communication from client to server. |
| [out] | server_crypto_wrapped_key | the crypto key during communication from server to client. |
| [in] | client_initial_vector | not use. |
| [in] | server_initial_vector | not use. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_VerifyDataGenerate()

fsp_err_t R_SCE_TLS_VerifyDataGenerate ( uint32_t *select_verify_data*, uint32_t * *sce_master_secret*, uint8_t * *hand_shake_hash*, uint8_t * *verify_data* )

Generate verify data.

**Parameters**

| | | |
|---|---|---|
| [in] | select_verify_data | Select Client/Server data. |
| [in] | sce_master_secret | master secret data. |
| [in] | hand_shake_hash | TLS hand shake message SHA256 HASH value. |
| [out] | verify_data | verify data. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_ServerKeyExchangeVerify()

fsp_err_t R_SCE_TLS_ServerKeyExchangeVerify ( uint32_t *public_key_type*, uint8_t * *client_random*, uint8_t * *server_random*, uint8_t * *server_ephemeral_ecdh_public_key*, uint8_t * *server_key_exchange_signature*, uint32_t * *encrypted_public_key*, uint32_t * *encrypted_ephemeral_ecdh_public_key* )

Retrives ECDH public key.

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [in] | server_ephemeral_ecdh_public_key | Ephemeral ECDH public key from Server. |
| [in] | server_key_exchange_signature | Server Key Exchange sigunature. |
| [in] | encrypted_public_key | encrypted public key. |
| [out] | encrypted_ephemeral_ecdh_public_key | encrypted Ephemeral ECDH public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1()

fsp_err_t R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1 ( uint32_t * *encrypted_public_key*, sce_tls_p256_ecc_wrapped_key_t * *tls_p256_ecc_wrapped_key*, uint32_t * *sce_pre_master_secret* )

Generate encrypted pre-master secret.

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_public_key | encrypted public key |
| [in] | tls_p256_ecc_wrapped_key | P-256 ECC key index. |
| [out] | sce_pre_master_secret | encrypted pre-master secret value for SCE. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### 5.2.15.9 Secure Crypto Engine (r_sce_protected_cavp)

Modules » Security

Driver for the CAVP Certified Secure Crypto Engine (SCE) on RA MCUs.

# Version

v1.1.0

The version that appears on the Components tab of the FSP Configuration editor is v1.1.0+fsp.<fsp_version>. The <fsp_version> metadata reflects tooling support files only and does not indicate any changes to the CAVP certified code. See Module Versioning for more information about component versioning in FSP.

# Functions

The specifications of this module comply with the Secure Crypto Engine (r_sce_protected) of FSP v5.5.0.

For more information about the features of this module, see the "Renesas Flexible Software Package (FSP) v5.5.0 User's Manual." The documentation can be obtained from the following link:

See also
https://www.renesas.com/en/software-tool/flexible-software-package-fsp

# Overview

This module provides SCE CAVP Certified functions.

### HW Overview

SCE9 Protected Mode CAVP certification has been obtained using the RA6M4 as the target processor.

| Crypto Peripheral version | Devices |
|---|---|
| SCE9 (Protected mode) | RA4M2, RA4M3, RA6M4, RA6M5 |

### Features

The SCE module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption
    - AES
      - ECB 128/256bit
      - CBC 128/256bit
      - GCM 128/256bit
      - CCM 128/256bit
  - Asymmetric Encryption/Decryption
    - RSA
      - RSAES-PKCS1-V1_5 1024/2048bit
      - RSAES-PKCS1-V1_5 3072/4096bit (Encryption only)
      - RSASSA-PKCS1-V1_5 1024/2048bit
      - RSASSA-PKCS1-V1_5 3072/4096bit (Verification only)
    - ECC
      - ECDSA secp192r1/secp224r1/secp256r1/secp384r1
      - ECDH secp256r1
  - Hash Functions
    - SHA-2
      - SHA-256
- Message Authentication Code
  - HMAC-SHA256bit
  - AES-CMAC 128/256bit
- Key Support
  - AES 128/256bit
  - AES Key Wrap/Key Unwrap 128/256bit
  - RSA 1024/2048bit
  - RSA 3072/4096bit (public key only)
  - ECC secp192r1/secp224r1/secp256r1/secp384r1
  - HMAC-SHA256bit
- TRNG
- TLS

- SSL / TLS support function (TLS1.2 compliant)

# Configuration

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Getting Started: Creating a SCE Protected Mode Project

Start by creating a new project in e² studio or RASC. On the Stacks tab, add New > Security > SCE Protected Mode(CAVP Certified). For information on how to install and update secure keys, refer to the Application Note R11AN0496.

### Reducing intialization time

The SCE intialization sequence can be modified to support a smaller subset of features and allow for a shorter intialization period. This is particularly useful in cases where startup time is important and the cryptographic features that are required are known in advance. The current fast boot configuration option supports the cryptographic primitives required by MCUBoot including:

- RSA 3K/4K signature verification
- ECC P256 signature verification
- SHA224/256 and GHASH calcuation

### Limitations

### Usage of R_SCE_ECDSA_secp384r1_SignatureGenerate/Verify

The SCE does not support SHA-384 in hardware, so the APIs listed below require the user to create a SHA-384 function for signature generation and verification. To use the APIs listed below, enable SCE_USER_SHA_384_ENABLED on RA Smart Configurator and prepare a function called SCE_USER_SHA_384_FUNCTION. The interface of SCE_USER_SHA_384_FUNCTION, which is called by the following APIs, is described below.

- R_SCE_ECDSA_secp384r1_SignatureGenerate()
- R_SCE_ECDSA_secp384r1_SignatureVerify()

### SCE_USER_SHA_384_FUNCTION()

```
uint32_t SCE_USER_SHA_384_FUNCTION(uint8_t * message, uint8_t * digest, uint32_t
message_length)
```

SHA-384 hash calculation is performed for an area extending the number of bytes specified by the argument message_length from the address specified by the argument message. The calculation result should be stored at the address specified by the argument digest.

Parameters

| | |
|---|---|
| message | [in] Start address of message |
| digest | [in,out] address for storing hash calculation result (48 bytes) |
| message_length | [in] Effective byte count of message |

Return values

| | |
|---|---|
| 0 | Hash value stored successfully. |
| others | Storing of hash value failed. |

# Examples

### AES Example

This is an example of AES-256 encryption and decryption.

```
static uint8_t plain[BLOCK * 2] =
{
    0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
    0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
};
void r_sce_example_aes ()
{
 sce_aes_handle_t      handle;
 sce_aes_wrapped_key_t wrapped_key;
    uint8_t               cipher_calculated[32] = {0};
    uint8_t               plain_calculated[32] = {0};
    uint32_t             dummy;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random key */
 R_SCE_AES256_WrappedKeyGenerate(&wrapped_key);
 /* Encrypt a plain text */
 R_SCE_AES256ECB_EncryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_EncryptUpdate(&handle, plain, cipher_calculated, BLOCK * 2);
```

```
 R_SCE_AES256ECB_EncryptFinal(&handle, cipher_calculated, &dummy);
 /* Decrypt a cipher text using same key as Encryption */
 R_SCE_AES256ECB_DecryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_DecryptUpdate(&handle, cipher_calculated, plain_calculated, BLOCK *
2);
 R_SCE_AES256ECB_DecryptFinal(&handle, plain_calculated, &dummy);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 /* Compare plain and plain_calculated */
 if (memcmp(plain, plain_calculated, BLOCK * 2))
    {
 while (1)
      {
 /* plain and plain_calculated are different (incorrect) */
       }
    }
 else
    {
 while (1)
      {
 /* plain and plain_calculated are the same (correct) */
       }
    }
}
```

### ECC Example

This is an example of ECC secp-256 signature generate and verify.

```
uint8_t ecc256_data_msg[] =
{
 's', 'a', 'm', 'p', 'l', 'e'
};
void r_sce_example_ecc ()
{
```

```
sce_ecc_wrapped_pair_key_t wrapped_pair_key;
sce_ecdsa_byte_data_t      message_hash =
   {
      0
   };
sce_ecdsa_byte_data_t signature;
   uint8_t               out_data[HW_SCE_ECDSA_DATA_BYTE_SIZE];
fsp_err_t             return_code;
   message_hash.data_length = sizeof(ecc256_data_msg);
   message_hash.pdata       = (uint8_t *) ecc256_data_msg;
   signature.pdata          = out_data;
/* SCE power on */
R_SCE_Open(&sce_ctrl, &sce_cfg);
/* Generate a random pair key */
R_SCE_ECC_secp256r1_WrappedKeyPairGenerate(&wrapped_pair_key);
/* Generate a Signature */
R_SCE_ECDSA_secp256r1_SignatureGenerate(&message_hash, &signature,
&wrapped_pair_key.priv_key);
/* Verify Signature and Public wrapped key */
   return_code = R_SCE_ECDSA_secp256r1_SignatureVerify(&signature, &message_hash,
&wrapped_pair_key.pub_key);
/* SCE power off */
R_SCE_Close(&sce_ctrl);
if (FSP_SUCCESS != return_code)
   {
while (1)
     {
/* Verify Fail */
      }
   }
else
   {
while (1)
     {
```

```
 /* Verify Success */
       }
    }
}
```

### RSA Example

This is an example of RSA-2048 encryption and decryption.

```
uint8_t rsa_msg[256] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c,
0x2d, 0x2e, 0x2f,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c,
0x3d, 0x3e, 0x3f,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c,
0x4d, 0x4e, 0x4f,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5a, 0x5b, 0x5c,
0x5d, 0x5e, 0x5f,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c,
0x6d, 0x6e, 0x6f,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7a, 0x7b, 0x7c,
0x7d, 0x7e, 0x7f,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c,
0x8d, 0x8e, 0x8f,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x9c,
0x9d, 0x9e, 0x9f,
    0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xab, 0xac,
0xad, 0xae, 0xaf,
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xbb, 0xbc,
0xbd, 0xbe, 0xbf,
```

```
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc,
0xcd, 0xce, 0xcf,
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xdb, 0xdc,
0xdd, 0xde, 0xdf,
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec,
0xed, 0xee, 0xef,
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff
};
void r_sce_example_rsa () {
 sce_rsa2048_wrapped_pair_key_t wrapped_pair_key;
 sce_rsa_byte_data_t            plain;
 sce_rsa_byte_data_t            plain_dec;
 sce_rsa_byte_data_t            cipher;
    uint8_t enc_cipher[HW_SCE_RSA_2048_DATA_BYTE_SIZE];
    uint8_t dec_plain[HW_SCE_RSA_2048_DATA_BYTE_SIZE];
    plain.data_length    = sizeof(rsa_msg) - PADDING_MINIMUM_BYTE_SIZE;
    plain.pdata          = (uint8_t *) rsa_msg;
    plain_dec.data_length = sizeof(dec_plain);
    plain_dec.pdata       = dec_plain;
    cipher.data_length    = sizeof(enc_cipher);
    cipher.pdata          = enc_cipher;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random key */
 R_SCE_RSA2048_WrappedKeyPairGenerate(&wrapped_pair_key);
 /* Encrypt a plain data */
 R_SCE_RSAES_PKCS2048_Encrypt(&plain, &cipher, &wrapped_pair_key.pub_key);
 /* Decrypt a plain data */
 R_SCE_RSAES_PKCS2048_Decrypt(&cipher, &plain_dec, &wrapped_pair_key.priv_key);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 /* Compare plain_dec and plain */
 if (0 != memcmp(plain_dec.pdata, plain.pdata, plain_dec.data_length))
```

```
    {
 while (1)
      {
 /* plain_dec and plain are different (incorrect) */
       }
    }
 else
    {
 while (1)
      {
 /* plain_dec and plain are the same (correct) */
       }
    }
}
```

### HASH Example

This is an example of calculating the SHA256 hash.

```
uint8_t message[] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f,
};
uint8_t hash[] =
{
    0x63, 0x0d, 0xcd, 0x29, 0x66, 0xc4, 0x33, 0x66, 0x91, 0x12, 0x54, 0x48, 0xbb,
0xb2, 0x5b, 0x4f,
    0xf4, 0x12, 0xa4, 0x9c, 0x73, 0x2d, 0xb2, 0xc8, 0xab, 0xc1, 0xb8, 0x58, 0x1b,
0xd7, 0x10, 0xdd
};
void r_sce_example_hash ()
{
```

```
sce_sha_md5_handle_t handle;
   uint8_t             digest[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE] = {0};
   uint32_t             digest_length = 0;
/* SCE power on */
R_SCE_Open(&sce_ctrl, &sce_cfg);
/* Encrypt a message text */
R_SCE_SHA256_Init(&handle);
R_SCE_SHA256_Update(&handle, &message, HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE);
R_SCE_SHA256_Final(&handle, &digest, &digest_length);
/* SCE power off */
R_SCE_Close(&sce_ctrl);
/* Compare digest and hash */
if (0 != memcmp(digest, hash, digest_length))
   {
while (1)
     {
/* digest and hash are different (incorrect) */
      }
   }
else
   {
while (1)
     {
/* digest and hash are the same (correct) */
      }
   }
}
```

### 5.2.15.10 Secure Key Injection (r_rsip_key_injection)

Modules » Security

#### Functions

| | |
|---|---|
| fsp_err_t | R_RSIP_AES128_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const |

| | |
|---|---|
| | p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_AES192_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_AES256_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_AES128_XTS_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_AES256_XTS_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ChaCha20_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_RSA2048_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_RSA3072_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_RSA4096_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const |

| | |
|---|---|
| | *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA224_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA256_InitialKeyWrap (rsip_key_injection_type_t |

| | |
|---|---|
| | const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA384_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA512_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA512_224_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| fsp_err_t | R_RSIP_HMAC_SHA512_256_InitialKeyWrap (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |

### Detailed Description

Driver for the Secure Key Injection on RA MCUs.

# Overview

This module provides RSIP functions in PSA Crypto.

### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| RSIP-E11A | RA4L1 |
| RSIP-E31A | RA4C1 |
| RSIP-E51A | RA8M1, RA8D1, RA8T1 |
| RSIP-E50D | RA8P1 |

### Features

The RSIP Key Injection Module has two types of APIs, InitialKeyWrap and EncryptedKeyWrap. The available APIs differ depending on used RSIP. Please refer to the following for details.

### Key Support

## AES Key Injection

| Crypto Peripheral version | 128 bit | 192 bit | 256 bit |
|---|---|---|---|
| RSIP-E11A | Support | No support | Support |
| RSIP-E31A | Support | No support | Support |
| RSIP-E51A | Support | Support | Support |
| RSIP-E50D | Support | Support | Support |

## AES XTS Key Injection

| Crypto Peripheral version | 128 bit XTS | 256 bit XTS |
|---|---|---|
| RSIP-E11A | No support | No support |
| RSIP-E31A | Support | Support |
| RSIP-E51A | Support | Support |
| RSIP-E50D | Support | Support |

## ChaCha20 Key Injection

| Crypto Peripheral version | ChaCha20 |
|---|---|
| RSIP-E11A | No support |
| RSIP-E31A | No support |
| RSIP-E51A | No support |
| RSIP-E50D | Support |

## RSA Key Injection

| Crypto Peripheral version | 2048 bit | 3072 bit | 4096 bit |
|---|---|---|---|
| RSIP-E11A | No support | No support | No support |
| RSIP-E31A | No support | No support | No support |
| RSIP-E51A | Support | Support | Support |
| RSIP-E50D | Support | Support | Support |

## ECC Key Injection

| Crypto Peripheral version | secp256r1 | secp384r1 | secp521r1 | secp256k1 | brainpoolP256r1 | brainpoolP384r1 | brainpoolP512r1 | Ed25519 |
|---|---|---|---|---|---|---|---|---|
| RSIP- | Support | No | No | Support | Support | No | No | No |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| E11A | | support | support | | | support | support | support |
| RSIP-E31A | Support | Support | No support | Support | Support | Support | No support | Support |
| RSIP-E51A | Support | Support | Support | Support | Support | Support | Support | Support |
| RSIP-E50D | Support | Support | Support | Support | Support | Support | Support | Support |

### HMAC Key Injection

| Crypto Peripheral version | SHA224 | SHA256 | SHA384 | SHA512 | SHA512/224 | SHA512/256 |
|---|---|---|---|---|---|---|
| RSIP-E11A | Support | Support | No support | No support | No support | No support |
| RSIP-E31A | No support | Support | Support | Support | Support | Support |
| RSIP-E51A | Support | Support | Support | Support | Support | Support |
| RSIP-E50D | Support | Support | Support | Support | Support | Support |

# Configuration

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Module activation

RSIP Key Umport for PSA Crypto Module is off by default. To use it, please On the Stacks tab, add New > Security > MbedTLS or MbedTLS (Crypto Only), Add Requires RSIP Driver > New > (*1), Add Key Injection for RSA Crypto (Optional) > New > Key Injectin for RSA Crypto for PSA Crypto Module after adding MbedTLS. For information on how to injection and update secure keys, refer to the Application Note R11AN0496.

### Hardware Initialization

Please refer to Hardware Initialization in Mbed Crypto H/W Acceleration (rm_psa_crypto).

### Enumerations

enum rsip_key_type_t

### Enumeration Type Documentation

### ◆ rsip_key_type_t

| enum rsip_key_type_t | |
|---|---|
| Enumerator | |
| RSIP_KEY_TYPE_INVALID | Invalid key. |
| RSIP_KEY_TYPE_AES_128 | AES-128. |
| RSIP_KEY_TYPE_AES_192 | AES-192. |
| RSIP_KEY_TYPE_AES_256 | AES-256. |
| RSIP_KEY_TYPE_XTS_AES_128 | XTS-AES-128. |
| RSIP_KEY_TYPE_XTS_AES_256 | XTS-AES-256. |
| RSIP_KEY_TYPE_CHACHA20 | ChaCha20. |
| RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC | secp256r1 public key (also known as NIST P-256, prime256v1) |
| RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC | secp384r1 public key (also known as NIST P-384) |
| RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC | secp521r1 public key (also known as NIST P-521) |
| RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC | secp256k1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC | brainpoolP256r1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC | brainpoolP384r1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC | brainpoolP512r1 public key |
| RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC | edwards25519 public key |
| RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE | secp256r1 private key (also known as NIST P-256, prime256v1) |
| RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE | secp384r1 private key (also known as NIST P-384) |
| RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE | secp521r1 private key (also known as NIST P-521) |
| RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE | secp256k1 private key |

| | |
|---|---|
| RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE | brainpoolP256r1 private key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE | brainpoolP384r1 private key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE | brainpoolP512r1 private key |
| RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE | edwards25519 private key |
| RSIP_KEY_TYPE_RSA_1024_PUBLIC | RSA-1024 public key. |
| RSIP_KEY_TYPE_RSA_2048_PUBLIC | RSA-2048 public key. |
| RSIP_KEY_TYPE_RSA_3072_PUBLIC | RSA-3072 public key. |
| RSIP_KEY_TYPE_RSA_4096_PUBLIC | RSA-4096 public key. |
| RSIP_KEY_TYPE_RSA_1024_PRIVATE | RSA-1024 private key. |
| RSIP_KEY_TYPE_RSA_2048_PRIVATE | RSA-2048 private key. |
| RSIP_KEY_TYPE_RSA_3072_PRIVATE | RSA-3072 private key. |
| RSIP_KEY_TYPE_RSA_4096_PRIVATE | RSA-4096 private key. |
| RSIP_KEY_TYPE_HMAC_SHA1 | HMAC-SHA1. |
| RSIP_KEY_TYPE_HMAC_SHA224 | HMAC-SHA224. |
| RSIP_KEY_TYPE_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_KEY_TYPE_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_KEY_TYPE_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_KEY_TYPE_HMAC_SHA512_224 | HMAC-SHA512/224. |
| RSIP_KEY_TYPE_HMAC_SHA512_256 | HMAC-SHA512/256. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_KEY_TYPE_KUK | Key Update Key (KUK) |

## Function Documentation

### ◆ R_RSIP_AES128_InitialKeyWrap()

fsp_err_t R_RSIP_AES128_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key* )

This API generates 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_AES192_InitialKeyWrap()

fsp_err_t R_RSIP_AES192_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key* )

This API generates 192-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 192-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_AES256_InitialKeyWrap()

fsp_err_t R_RSIP_AES256_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key* )

This API generates 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_AES128_XTS_InitialKeyWrap()

fsp_err_t R_RSIP_AES128_XTS_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key* )

This API generates 128-bit AES-XTS key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 128-bit AES-XTS wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_AES256_XTS_InitialKeyWrap()

fsp_err_t R_RSIP_AES256_XTS_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_aes_wrapped_key_t *const *p_wrapped_key* )

This API generates 256-bit AES-XTS key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit AES-XTS wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ChaCha20_InitialKeyWrap()

fsp_err_t R_RSIP_ChaCha20_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_chacha20_wrapped_key_t *const *p_wrapped_key* )

This API generates 256-bit ChaCha20-Poly-1305 key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit ChaCha20 wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_RSA2048_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA2048_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa2048_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 2048-bit RSA private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 2048-bit RSA wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_RSA3072_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA3072_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa3072_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 3072-bit RSA private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 3072-bit RSA wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_RSA4096_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_RSA4096_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_rsa4096_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 4096-bit RSA private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 4096-bit RSA wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp256r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  *p_wrapped_user_factory_programming_key*, uint8_t const *const  *p_initial_vector*, uint8_t const *const  *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*  )

This API generates 256-bit ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp384r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 384-bit ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 384-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp521r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  p_wrapped_user_factory_programming_key*, uint8_t const *const  p_initial_vector*, uint8_t const *const  p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*  )

This API generates 521-bit ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 521-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_secp256k1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  *p_wrapped_user_factory_programming_key*, uint8_t const *const  *p_initial_vector*, uint8_t const *const  *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*  )

This API generates 256-bit ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP256r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 256-bit brainpool ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP384r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 384-bit brainpool ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_brainpoolP512r1_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *p_initial_vector*, uint8_t const *const *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key* )

This API generates 512-bit brainpool ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 512-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap()

fsp_err_t R_RSIP_ECC_Ed25519_InitialPrivateKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  *p_wrapped_user_factory_programming_key*, uint8_t const *const  *p_initial_vector*, uint8_t const *const  *p_user_key*, rsip_ecc_private_wrapped_key_t *const *p_wrapped_key*  )

This API generates 256-bit Edwards curve ECC private key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA224_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA224_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key* )

This API generates 224-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 224-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA256_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA256_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key* )

This API generates 256-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 256-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA384_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA384_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key* )

This API generates 384-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 384-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA512_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const *const *p_wrapped_user_factory_programming_key*, uint8_t const *const *const *p_initial_vector*, uint8_t const *const *const *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key* )

This API generates 512-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 512-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA512_224_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_224_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  *p_wrapped_user_factory_programming_key*, uint8_t const *const  *p_initial_vector*, uint8_t const *const  *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*  )

This API generates 512/224-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 512/224-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

### ◆ R_RSIP_HMAC_SHA512_256_InitialKeyWrap()

fsp_err_t R_RSIP_HMAC_SHA512_256_InitialKeyWrap ( rsip_key_injection_type_t const *key_injection_type*, uint8_t const *const  *p_wrapped_user_factory_programming_key*, uint8_t const *const  *p_initial_vector*, uint8_t const *const  *p_user_key*, rsip_hmac_sha_wrapped_key_t *const *p_wrapped_key*  )

This API generates 512/256-bit HMAC-SHA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it is encrypted and MAC appended |
| [out] | p_wrapped_key | 512/256-bit HMAC-SHA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_ASSERTION A required parameter is NULL.
- FSP_ERR_CRYPTO_SCE_FAIL MAC anomaly detection.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT Resource conflict.
- FSP_ERR_CRYPTO_UNKNOWN An unknown error occurred.
- FSP_ERR_INVALID_STATE Internal state is illegal.
- FSP_ERR_UNSUPPORTED This API is not supported on this device.

***Note***

*The pre-run state is RSIP Enabled State. After the function runs the state transitions to RSIP Enabled State.*

## 5.2.15.11 Secure Key Injection (r_sce_key_injection)

Modules » Security

### Functions

| | |
|---|---|
| fsp_err_t | R_SCE_AES128_InitialKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_AES192_InitialKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_AES256_InitialKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_KeyUpdateKeyWrap (const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_key_update_key_t *const key_update_key) |
| fsp_err_t | R_SCE_AES128_EncryptedKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_AES192_EncryptedKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_AES256_EncryptedKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA3072_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa3072_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA4096_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, |

| | |
|---|---|
| | sce_rsa4096_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_RSA2048_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256k1_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256k1_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |

| | |
|---|---|
| fsp_err_t | R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256k1_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp256k1_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP256r1_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP256r1_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP384r1_InitialPublicKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP384r1_InitialPrivateKeyWrap (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, |

| | |
|---|---|
| | sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP256r1_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP256r1_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP384r1_EncryptedPublicKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| fsp_err_t | R_SCE_ECC_brainpoolP384r1_EncryptedPrivateKeyWrap (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |

### Detailed Description

Driver for the Secure Key Injection on RA MCUs.

# Overview

This module provides SCE functions in PSA Crypto.

### HW Overview

| Crypto Peripheral version | Devices |
|---|---|
| SCE9 | RA6M5, RA6M4, RA4M3, RA4M2 |
| SCE7 | RA6M3, RA6M2, RA6M1, RA6T1 |
| SCE5 | RA4W1, RA4M1 |
| SCE5B | RA6T2 |

### Features

The SCE Key Injection Module has two types of APIs, InitialKeyWrap and EncryptedKeyWrap. The available APIs differ depending on used SCE. Please refer to the following for details.

- Key Support
  - AES 128/256bit

| Crypto Peripheral version | 128 bit | 192 bit | 256 bit |
|---|---|---|---|
| SCE9 | InitialKeyWrap Only | InitialKeyWrap Only | InitialKeyWrap Only |

| | | | |
|---|---|---|---|
| SCE7 | Support | Support | Support |
| SCE5 | Support | No support | Support |
| SCE5B | InitialKeyWrap Only | No support | InitialKeyWrap Only |

- RSA 2048/3072/4096bit

| Crypto Peripheral version | 2048 bit | 3072 bit (public) | 4096 bit (public) |
|---|---|---|---|
| SCE9 | InitialKeyWrap Only | InitialKeyWrap Only | InitialKeyWrap Only |
| SCE7 | Support | No support | No support |
| SCE5 | No support | No support | No support |
| SCE5B | No support | No support | No support |

- ECC secp256r1/secp256k1/secp384r1

| Crypto Peripheral version | secp256r1 | secp256k1 | secp384r1 | secp256r1 Brainpool | secp384r1 Brainpool |
|---|---|---|---|---|---|
| SCE9 | InitialKeyWrap Only | InitialKeyWrap Only | InitialKeyWrap Only | InitialKeyWrap Only | InitialKeyWrap Only |
| SCE7 | Support | Support | Support | Support | Support |
| SCE5 | No support | No support | No support | No support | No support |
| SCE5B | No support | No support | No support | No support | No support |

# Configuration

### Clock Configuration

This module does not require a specific clock configuration.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Module activation

SCE Key Injection for PSA Crypto Module is off by default. To use it, please On the Stacks tab, add New > Security > MbedTLS or MbedTLS (Crypto Only), Add Requires SCE Driver > New > (*1), Add Key Injection for RSA Crypto (Optional) > New > Key Injection for RSA Crypto for PSA Crypto Module after adding MbedTLS. For information on how to import and update secure keys, refer to the Application Note R11AN0496.
(*1)

| | |
|---|---|

| Crypto Peripheral version | String displayed in the Stacks tab |
|---|---|
| SCE9 | SCE Compatibility Mode |
| SCE7 | SCE7 |
| SCE5 | SCE5 |
| SCE5B | SCE5B |

### Hardware Initialization

Please refer to Hardware Initialization in Mbed Crypto H/W Acceleration (rm_psa_crypto).

### Function Documentation

### ◆ R_SCE_AES128_InitialKeyWrap()

fsp_err_t R_SCE_AES128_InitialKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API generates 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES192_InitialKeyWrap()

fsp_err_t R_SCE_AES192_InitialKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API generates 192-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 192-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256_InitialKeyWrap()

fsp_err_t R_SCE_AES256_InitialKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_KeyUpdateKeyWrap()

fsp_err_t R_SCE_KeyUpdateKeyWrap ( const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_key_update_key_t *const *key_update_key* )

This API generates a key update key which is used for functions of the key updating.

**Parameters**

| | | |
|---|---|---|
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | key_update_key | Key update key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES128_EncryptedKeyWrap()

fsp_err_t R_SCE_AES128_EncryptedKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API wraps 128-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES192_EncryptedKeyWrap()

fsp_err_t R_SCE_AES192_EncryptedKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API wraps 192-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 192-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_AES256_EncryptedKeyWrap()

fsp_err_t R_SCE_AES256_EncryptedKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_aes_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit AES key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_InitialPublicKeyWrap()

fsp_err_t R_SCE_RSA2048_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_rsa2048_public_wrapped_key_t *const *wrapped_key* )

This API generates 2048-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA3072_InitialPublicKeyWrap()

fsp_err_t R_SCE_RSA3072_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_rsa3072_public_wrapped_key_t *const *wrapped_key* )

This API generates 3072-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 3072-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA4096_InitialPublicKeyWrap()

fsp_err_t R_SCE_RSA4096_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_rsa4096_public_wrapped_key_t *const *wrapped_key* )

This API generates 4096-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 4096-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_InitialPrivateKeyWrap()

fsp_err_t R_SCE_RSA2048_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_rsa2048_private_wrapped_key_t *const *wrapped_key* )

This API generates 2048-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_rsa2048_public_wrapped_key_t *const *wrapped_key* )

This API wraps 2048-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_RSA2048_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_rsa2048_private_wrapped_key_t *const *wrapped_key* )

This API wraps 2048-bit RSA key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_InitialPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256k1_InitialPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256k1_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_InitialPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API generates 384-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 364-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_InitialPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256k1_InitialPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256k1_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_InitialPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API generates 384-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256k1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256k1_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API wraps 384-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp256k1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256k1_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API wraps 384-bit ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP256r1_InitialPublicKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP256r1_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP256r1_InitialPrivateKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP256r1_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API generates 256-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP384r1_InitialPublicKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP384r1_InitialPublicKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API generates 384-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 364-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP384r1_InitialPrivateKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP384r1_InitialPrivateKeyWrap ( const uint8_t *const *key_type*, const uint8_t *const *wrapped_user_factory_programming_key*, const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API generates 384-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Selection key type when generating wrapped key (0: for encrypted key, 1: for plain key) |
| [in] | wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | initial_vector | Initialization vector when generating encrypted_key. When key_type is 1 as plain key, this is not required and any value can be specified. |
| [in] | encrypted_key | Encrypted user key and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP256r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP256r1_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP256r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP256r1_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API wraps 256-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP384r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP384r1_EncryptedPublicKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_public_wrapped_key_t *const *wrapped_key* )

This API wraps 384-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.

- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

### ◆ R_SCE_ECC_brainpoolP384r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_brainpoolP384r1_EncryptedPrivateKeyWrap ( const uint8_t *const *initial_vector*, const uint8_t *const *encrypted_key*, const sce_key_update_key_t *const *key_update_key*, sce_ecc_private_wrapped_key_t *const *wrapped_key* )

This API wraps 384-bit Brainpool ECC key within the user routine.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination. |
| FSP_ERR_UNSUPPORTED | API not supported. |

**Returns**

If an error occurs, the return value will be as follows.
- FSP_ERR_CRYPTO_SCE_FAIL Internal I/O buffer is not empty.
- FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT A resource conflict occurred because a hardware resource needed.

***Note***

*The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.*

## 5.2.15.12 TinyCrypt H/W Acceleration (rm_tinycrypt_port)

Modules » Security

**Functions**

| | |
|---|---|
| fsp_err_t | RM_TINCYRYPT_PORT_Init () |
| fsp_err_t | RM_TINCYRYPT_PORT_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes)<br>Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer. More... |
| int | default_CSPRNG (uint8_t *dest, unsigned int size)<br>Implements the Cryptographically Secure Pseudo-Random Number Generator function required byt TinyCrypt. More... |

### Detailed Description

AES128 Hardware acceleration for TinyCrypt on the RA2 family.

# Overview

*Note*

***The TinyCrypt port module does not provide any interfaces to the user. Consult the documentation at https://github.com/intel/tinycrypt/blob/master/documentation/tinycrypt.rst for further information.***

TinyCrypt is designed as a small footprint software crypto implementation to be used on resource constrained devices. The software only module is available in FSP on all RA devices. Hardware acceleration for AES-128 is provided only for the RA2 family. This release uses TinyCrypt v0.2.8.

### Hardware Overview

| Crypto Peripheral version | Devices |
|---|---|
| AES Engine | RA2A1, RA2E1, RA2L1 |

### Features

For features supported by the software-only version, refer to the TinyCrypt documentation.

The TinyCrypt port module provides hardware support for the following operations

- AES
  - Keybits - 128, 192, 256
  - ECB, CBC, CTR, CCM, GCM and CMAC modes  -TRNG

# Configuration

### Build Time Configurations for TinyCrypt_Acceleration

The following build time configurations are defined in fsp_cfg/rm_tinycrypt_port_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### AES Configuration

To enable hardware acceleration for the AES128 operation, choose TinyCrypt (H/W Accelerated) from the stack options . This feature is only supported on the RA2 family.

# Usage Notes

### Hardware Initialization

Invoke RM_TINCYRYPT_PORT_Init () to initialize the hardware before using Tinycrypt if either

hardware acceleration or the TRNG is to be used.

### Random Number Generation

There are two Pseudo-random Number Generators (PRNG) provided in TinyCrypt

- CTR-PRNG which uses AES128 internally in its implementation. Enabling AES128 hardware acceleration will improve the performance of this module.
- HMAC-PRNG which uses SHA256 internally in its implementation.

**Both these implementations will only be able to provide a random pseudo-random number sequence if they are seeded with truly random data. The TRNG module that is present in hardware and available in rm_tinycrypt_port must be used to seed these modules.** When using CTR-PRNG or HMAC-PRNG, use the RM_TINCYRYPT_PORT_TRNG_Read() function to obtain random data from the TRNG hardware and use that to seed the PRNG modules before invoking the pseudo-random number generation. If purely random data is sufficient for the application, then RM_TINCYRYPT_PORT_TRNG_Read() can be used directly instead. The hardware TRNG implements the CTR_DRBG spec.

### Default CSPRNG

The TinyCrypt ECC implementation requires a platform specific implementation of the default_CSPRNG() function. This function has been implemented using the hardware TRNG in the port to support software ECC usage. When using TinyCrypt in S/W mode, it is necessary to implement default_CSPRNG() if using ECC signature generation (ECDSA) or key derivation (ECDH).

### AES-128 Usage

The AES ECB mode implementation is provided in aes_encrypt.decrypt.c. All the other modes of AES operation including CBC, CCN, CMAC and CTR use the ECB mode for the block operation. On the RA2, the ECB mode has been hardware accelerated which improves performance of the other modes as well. Additionally the CBC and CTR modes are also accelerated.

To use the different AES modes, first initialize the hardware (on the RA2) and then use the functions defined in the header file of each AES mode. Note that TinyCrypt does not provide any type of padding or buffering so the data provided to these modes should be multiples of AES block size.

### Usage with RA2A2

On the RA2A2, the TinyCrypt API has been expanded to support GCM mode operation. Key lengths of 128, 192 and 256 are supported. GCM mode operation is not supported on other MCUs in hardware or software in the TinyCrypt API.

### Memory Usage

TinyCrypt does not use dynamic allocation so there is no heap requirement.

### Limitations

### Usage with RA4 and RA6 devices

TinyCrypt (S/W Only) can be used on RA4 and RA6 devices. However, since ECC signature generation (ECDSA) and key derivation (ECDH) requires a random number source, that operation is currently not supported on these devices when using TinyCrypt (S/W Only). In order to support those operations the function default_CSPRNG() must be implemented in the user code.

### TinyCrypt

- No padding is supported; the user is expected to provide adequately padded data depending on the algorithm used.
- AES Key generation is not supported.
- Key encoding/decoding is not supported.

### Using TinyCrypt with TrustZone

Unlike FSP drivers, TinyCrypt cannot be configured as Non-secure callable in the RA Configurator for a secure project. The reason for this is that in order to achieve the security objective of controlling access to protected keys, both the crypto code as well as the keys must be placed in the secure region. Since the tinyCrypt API requires access to the keys directly during initialization and later via a key handle, allowing non-secure code to use the API by making it Non-secure callable will require the keys to be stored in non-secure memory.

This limitation is identical to that for PSA Crypto. Refer to the documentation of that module on how to create a crypto Non-Secure Callable layer to be used in such situations.

## Examples

### AES-CBC Example

This is an example on using TinyCrypt to encrypt and decrypt data using an AES-128 key in CBC mode.

```
#define TC_INPUT_PLAINTEXT_SIZE 64U
#define TF_AES_IV_SIZE TC_AES_BLOCK_SIZE
#define TC_OUTPUT_CIPHERTEXT_SIZE (TC_INPUT_PLAINTEXT_SIZE + TF_AES_IV_SIZE)
/*
 * NIST test vectors from SP 800-38a:
 *
 * Block #1
 * Plaintext 6bc1bee22e409f96e93d7e117393172a
 * Input Block 6bc0bce12a459991e134741a7f9e1925
 * Output Block 7649abac8119b246cee98e9b12e9197d
 * Ciphertext 7649abac8119b246cee98e9b12e9197d
 * Block #2
 * Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
 * Input Block d86421fb9f1a1eda505ee1375746972c
 * Output Block 5086cb9b507219ee95db113a917678b2
 * Ciphertext 5086cb9b507219ee95db113a917678b2
 * Block #3
 * Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
```

```
 * Input Block 604ed7ddf32efdff7020d0238b7c2a5d
 * Output Block 73bed6b8e3c1743b7116e69e22229516
 * Ciphertext 73bed6b8e3c1743b7116e69e22229516
 * Block #4
 * Plaintext f69f2445df4f9b17ad2b417be66c3710
 * Input Block 8521f2fd3c8eef2cdc3da7e5c44ea206
 * Output Block 3ff1caa1681fac09120eca307586e1a7
 * Ciphertext 3ff1caa1681fac09120eca307586e1a7
 */
const uint8_t cbc_key[TC_AES_KEY_SIZE] =
{
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09,
0xcf, 0x4f, 0x3c
};
uint8_t cbc_iv[TC_AES_BLOCK_SIZE] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f
};
const uint8_t cbc_plaintext[TC_INPUT_PLAINTEXT_SIZE] =
{
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73,
0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45,
0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a,
0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6,
0x6c, 0x37, 0x10
};
uint8_t cbc_expected_ciphertext[TC_OUTPUT_CIPHERTEXT_SIZE] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f, // NOLINT(readability-magic-numbers)
```

```
    0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46, 0xce, 0xe9, 0x8e, 0x9b, 0x12,
0xe9, 0x19, 0x7d, // NOLINT(readability-magic-numbers)
    0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0xee, 0x95, 0xdb, 0x11, 0x3a, 0x91,
0x76, 0x78, 0xb2, // NOLINT(readability-magic-numbers)
    0x73, 0xbe, 0xd6, 0xb8, 0xe3, 0xc1, 0x74, 0x3b, 0x71, 0x16, 0xe6, 0x9e, 0x22,
0x22, 0x95, 0x16, // NOLINT(readability-magic-numbers)
    0x3f, 0xf1, 0xca, 0xa1, 0x68, 0x1f, 0xac, 0x09, 0x12, 0x0e, 0xca, 0x30, 0x75,
0x86, 0xe1, 0xa7 // NOLINT(readability-magic-numbers)
};
void tinycrypt_aes128cbc_example (void)
{
 struct tc_aes_key_sched_struct aes_keyschedule;
    uint8_t cbc_encrypted[TC_OUTPUT_CIPHERTEXT_SIZE] = {0U};
    uint8_t cbc_decrypted[TC_OUTPUT_CIPHERTEXT_SIZE] = {0U};
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&aes_keyschedule, cbc_key))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_cbc_mode_encrypt(cbc_encrypted, sizeof(cbc_plaintext) +
TC_AES_BLOCK_SIZE, cbc_plaintext,
 sizeof(cbc_plaintext), cbc_iv, &aes_keyschedule))
    {
       debugger_break();
    }
 else if (0 != memcmp(&cbc_encrypted[0], &cbc_expected_ciphertext[0], sizeof
(cbc_encrypted)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_cbc_mode_decrypt(cbc_decrypted, sizeof(cbc_encrypted),
&cbc_encrypted[TC_AES_BLOCK_SIZE],
 sizeof(cbc_encrypted), cbc_encrypted, &aes_keyschedule))
```

```
    {
       debugger_break();
    }
 else if (0 != memcmp(&cbc_plaintext[0], &cbc_decrypted[0], sizeof(cbc_plaintext)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
 while (1)
     {
           ;
     }
    }
}
```

### AES-CTR Example

This is an example on using TinyCrypt to encrypt and decrypt data using an AES-128 key in CTR mode.

```
#define TC_CTR_INPUT_PLAINTEXT_SIZE 64U
#define TF_AES_IV_SIZE TC_AES_BLOCK_SIZE
#define TC_CTR_OUTPUT_CIPHERTEXT_SIZE (TC_CTR_INPUT_PLAINTEXT_SIZE + TF_AES_IV_SIZE)
/*
 * NIST SP 800-38a CTR Test for encryption and decryption.
 */
const uint8_t ctr_key[TC_AES_KEY_SIZE] =
{
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09,
0xcf, 0x4f, 0x3c
};
uint8_t ctr_iv[TC_AES_KEY_SIZE] =
{
```

```
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff // NOLINT(readability-magic-numbers)
};
const uint8_t ctr_plaintext[64] =
{
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73,
0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45,
0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a,
0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6,
0x6c, 0x37, 0x10
};
const uint8_t ctr_expected_ciphertext[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] =
{
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff,
    0x87, 0x4d, 0x61, 0x91, 0xb6, 0x20, 0xe3, 0x26, 0x1b, 0xef, 0x68, 0x64, 0x99,
0x0d, 0xb6, 0xce,
    0x98, 0x06, 0xf6, 0x6b, 0x79, 0x70, 0xfd, 0xff, 0x86, 0x17, 0x18, 0x7b, 0xb9,
0xff, 0xfd, 0xff,
    0x5a, 0xe4, 0xdf, 0x3e, 0xdb, 0xd5, 0xd3, 0x5e, 0x5b, 0x4f, 0x09, 0x02, 0x0d,
0xb0, 0x3e, 0xab,
    0x1e, 0x03, 0x1d, 0xda, 0x2f, 0xbe, 0x03, 0xd1, 0x79, 0x21, 0x70, 0xa0, 0xf3,
0x00, 0x9c, 0xee
};
void tinycrypt_aes128ctr_example (void)
{
 struct tc_aes_key_sched_struct aes_keyschedule;
    uint8_t ctr_encrypted[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] = {0U};
    uint8_t ctr_decrypted[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] = {0U};
 if (0 != memcpy(ctr_encrypted, ctr_iv, sizeof(ctr_iv)))
    {
```

```
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&aes_keyschedule, ctr_key))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ctr_mode(&ctr_encrypted[TC_AES_BLOCK_SIZE], sizeof(ctr_plaintext),
ctr_plaintext, sizeof(ctr_plaintext),
                         ctr_iv, &aes_keyschedule))
    {
       debugger_break();
    }
 else if (0 != memcmp(&ctr_encrypted[0], &ctr_expected_ciphertext[0], sizeof
(ctr_encrypted)))
    {
       debugger_break();
    }
 else if (0 != memcpy(ctr_iv, ctr_encrypted, sizeof(ctr_iv)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ctr_mode(ctr_decrypted, sizeof(ctr_decrypted),
&ctr_encrypted[TC_AES_BLOCK_SIZE], sizeof(ctr_decrypted),
                         ctr_iv, &aes_keyschedule))
    {
       debugger_break();
    }
 else if (0 != memcmp(&ctr_plaintext[0], &ctr_decrypted[0], sizeof(ctr_plaintext)))
    {
       debugger_break();
    }
 else
```

```
    {
 /* Operation successful. */
 while (1)
      {
            ;
      }
    }
}
```

### CTR-PRNG Example

This is an example on using the CTR_PRNG module in TinyCrypt to obtain random data.

```
#define TC_ENTROPY_SIZE 64U
#define TC_CTRPRNG_OUTPUT_SIZE 32U
void tinycrypt_ctr_prng_example (void)
{
    TCCtrPrng_t cprng_ctx;
    uint8_t    seed[TC_ENTROPY_SIZE];
    uint8_t    ctr_prng_output_1[TC_CTRPRNG_OUTPUT_SIZE] = {0};
    uint8_t    ctr_prng_output_2[TC_CTRPRNG_OUTPUT_SIZE] = {0};
 /* Setup the platform; initialize the crypto engine. */
 if (0 != RM_TINCYRYPT_PORT_Init())
    {
       debugger_break();
    }
 /* Read random data from the TRNG to use as seed for the CTR_PRNG. */
 else if (FSP_SUCCESS != RM_TINCYRYPT_PORT_TRNG_Read(seed, sizeof(seed)))
    {
       debugger_break();
    }
 /* Initialize and seed the CTR_PRNG with the random data from the TRNG. */
 else if (TC_CRYPTO_SUCCESS != tc_ctr_prng_init(&cprng_ctx, seed, sizeof(seed), 0,
0))
    {
```

```
      debugger_break();
   }
 /* Read random data from the CTR_PRNG. */
 else if (TC_CRYPTO_SUCCESS !=
            tc_ctr_prng_generate(&cprng_ctx, 0, 0, ctr_prng_output_1, sizeof
(ctr_prng_output_1)))
   {
      debugger_break();
   }
 /* Check that the generated value is not 0. */
 else if (0 == memcmp(&ctr_prng_output_1[0], &ctr_prng_output_2[0], sizeof
(ctr_prng_output_1)))
   {
      debugger_break();
   }
 /* Read random data again from the TRNG. */
 else if (TC_CRYPTO_SUCCESS !=
            tc_ctr_prng_generate(&cprng_ctx, 0, 0, ctr_prng_output_2, sizeof
(ctr_prng_output_2)))
   {
      debugger_break();
   }
 /* Check that the generated value is different than the previous call. */
 else if (0 == memcmp(&ctr_prng_output_1[0], &ctr_prng_output_2[0], sizeof
(ctr_prng_output_1)))
   {
      debugger_break();
   }
 else
   {
 /* Operation successful. */
 while (1)
     {
           ;
```

```
      }
    }
}
```

### AES-CMAC Example

This is an example on using AES Circuit (Hardware support) to encrypt data using an AES-128 key in CMAC mode. This is also an example on use tc_cmac_setup and tc_cmac_setup_extended.

```
#define MLEN4 64
#define BUF_LEN 16
const uint8_t cmac_key_128[TC_AES_KEY_SIZE] =
{
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
    0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c
};
const uint8_t msg[MLEN4] =
{
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
    0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
    0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11,
    0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17,
    0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10,
};
const uint8_t tag[BUF_LEN] =
{
    0x51, 0xf0, 0xbe, 0xbf, 0x7e, 0x3b, 0x9d, 0x92,
    0xfc, 0x49, 0x74, 0x17, 0x79, 0x36, 0x3c, 0xfe,
};
void tinycrypt_aes128cmac_example (void)
{
 struct tc_cmac_struct          state;
```

```
struct tc_aes_key_sched_struct sched;
   uint8_t Tag[BUF_LEN];
if (TC_CRYPTO_SUCCESS != tc_cmac_setup(&state, cmac_key_128, &sched))
   {
      debugger_break();
   }
else if (TC_CRYPTO_SUCCESS != tc_cmac_init(&state))
   {
      debugger_break();
   }
else if (TC_CRYPTO_SUCCESS != tc_cmac_update(&state, msg, sizeof(msg)))
   {
      debugger_break();
   }
else if (TC_CRYPTO_SUCCESS != tc_cmac_final(Tag, &state))
   {
      debugger_break();
   }
else if (TC_CRYPTO_SUCCESS != memcmp(Tag, tag, BUF_LEN))
   {
      debugger_break();
   }
else
   {
/* Operation successful. */
while (1)
     {
           ;
     }
   }
}
/* Below is a example of using tc_cmac_setup_extended instead of tc_cmac_setup*/
void tinycrypt_aes128cmac_example_setup_extended (void)
{
```

```
 struct tc_cmac_struct          state;
 struct tc_aes_key_sched_struct sched;
    uint8_t Tag[BUF_LEN];
 if (TC_CRYPTO_SUCCESS != tc_cmac_setup_extended(&state, cmac_key_128, &sched,
sizeof(cmac_key_128)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != tc_cmac_init(&state))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != tc_cmac_update(&state, msg, sizeof(msg)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != tc_cmac_final(Tag, &state))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != memcmp(Tag, tag, BUF_LEN))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
 while (1)
      {
            ;
      }
    }
}
```

### AES-CCM Example

This is an example on using AES Circuit (Hardware support) to encrypt data using an AES-128 key in CCM mode. This is also an example on use tc_aes128_set_encrypt_key and tc_aes_set_encrypt_key_extended.

```
#define NONCE_LEN7 (7)
#define M_LEN16 (16)
#define NUM_NIST_KEYS128 (16)
#define DATA_BUF_LEN24 (24)
#define HEADER_LEN32 (32)
#define ENCRYPTED_BUF_LEN40 (40)
#define TC_CCM_MAX_CT_SIZE (50)
const uint8_t key[NUM_NIST_KEYS128] =
{
    0xc0, 0x42, 0x5e, 0xd2, 0x0c, 0xd2, 0x8f, 0xda, // NOLINT(readability-magic-
numbers)
    0x67, 0xa2, 0xbc, 0xc0, 0xab, 0x34, 0x2a, 0x49 // NOLINT(readability-magic-
numbers)
};
uint8_t nonce[NONCE_LEN7] =
{
    0x37, 0x66, 0x7f, 0x33, 0x4d, 0xce, 0x90       // NOLINT(readability-magic-
numbers)
};
const uint8_t hdr[HEADER_LEN32] =
{
    0x0b, 0x3e, 0x8d, 0x97, 0x85, 0xc7, 0x4c, 0x8f, // NOLINT(readability-magic-
numbers)
    0x41, 0xea, 0x25, 0x7d, 0x4d, 0x87, 0x49, 0x5f, // NOLINT(readability-magic-
numbers)
    0xfb, 0xbb, 0x33, 0x55, 0x42, 0xb1, 0x2e, 0x0d, // NOLINT(readability-magic-
numbers)
    0x62, 0xbb, 0x17, 0x7e, 0xc7, 0xa1, 0x64, 0xd9 // NOLINT(readability-magic-
numbers)
};
```

```
const uint8_t data[DATA_BUF_LEN24] =
{
    0x4f, 0x06, 0x5a, 0x23, 0xee, 0xca, 0x6b, 0x18, // NOLINT(readability-magic-
numbers)
    0xd1, 0x18, 0xe1, 0xde, 0x4d, 0x7e, 0x5c, 0xa1, // NOLINT(readability-magic-
numbers)
    0xa7, 0xc0, 0xe5, 0x56, 0xd7, 0x86, 0xd4, 0x07 // NOLINT(readability-magic-
numbers)
};
const uint8_t expected[ENCRYPTED_BUF_LEN40] =
{
    0x76, 0x8f, 0xcc, 0xdf, 0x48, 0x98, 0xbc, 0xa0, // NOLINT(readability-magic-
numbers)
    0x99, 0xe3, 0x3c, 0x3d, 0x40, 0x56, 0x54, 0x97, // NOLINT(readability-magic-
numbers)
    0xde, 0xc2, 0x2d, 0xd6, 0xe3, 0x3d, 0xcf, 0x43, // NOLINT(readability-magic-
numbers)
    0x84, 0xd7, 0x1b, 0xe8, 0x56, 0x5c, 0x21, 0xa4, // NOLINT(readability-magic-
numbers)
    0x55, 0xdb, 0x45, 0x81, 0x6d, 0xa8, 0x15, 0x8c // NOLINT(readability-magic-
numbers)
};
void tinycrypt_aes128ccm_example (void)
{
 struct tc_ccm_mode_struct      c;
 struct tc_aes_key_sched_struct sched;
    uint8_t encrypted_data[TC_CCM_MAX_CT_SIZE] = {0};
    uint8_t decrypted_data[DATA_BUF_LEN24]    = {0};
/* Encryption */
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&sched, key))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_ccm_config(&c, &sched, nonce, sizeof(nonce),
```

```
M_LEN16))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ccm_generation_encryption(encrypted_data, sizeof(encrypted_data),
hdr, sizeof(hdr), data, sizeof(data),
                                          &c))
    {
       debugger_break();
    }
 else if (0 != memcmp(encrypted_data, expected, sizeof(expected)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
    }
/* Decryption */
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_decrypt_key(&sched, key))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_ccm_config(&c, &sched, nonce, sizeof(nonce),
M_LEN16))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ccm_decryption_verification(decrypted_data, sizeof(decrypted_data),
hdr, sizeof(hdr), expected,
 sizeof(expected), &c))
    {
```

```
       debugger_break();
    }
 else if (0 != memcmp(decrypted_data, data, sizeof(data)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
    }
}
void tinycrypt_aes128ccm_example_key_extended (void)
{
 struct tc_ccm_mode_struct      c;
 struct tc_aes_key_sched_struct sched;
    uint8_t encrypted_data[TC_CCM_MAX_CT_SIZE] = {0};
    uint8_t decrypted_data[DATA_BUF_LEN24]    = {0};
/* Encryption */
 if (TC_CRYPTO_SUCCESS != tc_aes_set_encrypt_key_extended(&sched, key, sizeof(key)))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_ccm_config(&c, &sched, nonce, sizeof(nonce),
M_LEN16))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ccm_generation_encryption(encrypted_data, sizeof(encrypted_data),
hdr, sizeof(hdr), data, sizeof(data),
                                          &c))
    {
       debugger_break();
    }
```

```
 else if (0 != memcmp(encrypted_data, expected, sizeof(expected)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
    }
/* Decryption */
 if (TC_CRYPTO_SUCCESS != tc_aes_set_decrypt_key_extended(&sched, key, sizeof(key)))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_ccm_config(&c, &sched, nonce, sizeof(nonce),
M_LEN16))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ccm_decryption_verification(decrypted_data, sizeof(decrypted_data),
hdr, sizeof(hdr), expected,
 sizeof(expected), &c))
    {
       debugger_break();
    }
 else if (0 != memcmp(decrypted_data, data, sizeof(data)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
    }
}
```

### AES-GCM Example

This is an example on using AES Circuit (Hardware support) to encrypt data using an AES-128 key in GCM mode. This operation is supported only on the RA2A2 device.

```
/*
 * gcmEncryptExtIV128.rsp AES-GCM Test for encryption.
 *
 * # CAVS 14.0
 * # GCM Encrypt with keysize 128 test information
 * # Generated on Fri Aug 31 11:23:06 2012
 * [Keylen = 128]
 * [IVlen = 1024]
 * [PTlen = 408]
 * [AADlen = 720]
 * [Taglen = 32]
 * Count = 14
 * Key = 0e00c76561d2bd9b40c3c15427e2b08f
 * IV = 492cadaccd3ca3fbc9cf9f06eb3325c4e159850b0dbe98199b89b7af528806610b6f63998e1ea
e80c348e74cbb921d8326631631fc6a5d304f39166daf7ea15fa1977f101819adb510b50fe9932e12c5a8
5aa3fd1e73d8d760af218be829903a77c63359d75edd91b4f6ed5465a72662f5055999e059e7654a8edc9
21aa0d496
 * PT = fef03c2d7fb15bf0d2df18007d99f967c878ad59359034f7bb2c19af120685d78e32f6b8b83b0
32019956ca9c0195721476b85
 * AAD = d8f1163d8c840292a2b2dacf4ac7c36aff8733f18fabb4fa5594544125e03d1e6e5d6d0fd616
56c8d8f327c92839ae5539bb469c9257f109ebff85aad7bd220fdaa95c022dbd0c7bb2d878ad504122c94
3045d3c5eba8f1f56c0
 * CT = 4f6cf471be7cbd2575cd5a1747aea8fe9dea83e51936beac3e68f66206922060c697ffa7af80a
d6bb68f2cf4fc97416ee52abe
 * Tag = e20b6655
 */
#define IV_LEN_IN_BYTE 128U
#define PT_LEN_IN_BYTE 51U
#define AAD_LEN_IN_BYTE 90U
```

```
#define TAG_LEN_IN_BYTE 4U
#define KEYLEN_128BIT 128U
uint8_t gcm_key_128[TC_AES_KEY_SIZE] =
{
    0x0e, 0x00, 0xc7, 0x65, 0x61, 0xd2, 0xbd, 0x9b, 0x40, 0xc3, 0xc1, 0x54, 0x27,
0xe2, 0xb0, 0x8f // NOLINT(readability-magic-numbers)
};
uint8_t IV[IV_LEN_IN_BYTE] =
{
    0x49, 0x2c, 0xad, 0xac, 0xcd, 0x3c, 0xa3, 0xfb, 0xc9, 0xcf, 0x9f, 0x06, 0xeb,
0x33, 0x25, 0xc4, // NOLINT(readability-magic-numbers)
    0xe1, 0x59, 0x85, 0x0b, 0x0d, 0xbe, 0x98, 0x19, 0x9b, 0x89, 0xb7, 0xaf, 0x52,
0x88, 0x06, 0x61, // NOLINT(readability-magic-numbers)
    0x0b, 0x6f, 0x63, 0x99, 0x8e, 0x1e, 0xae, 0x80, 0xc3, 0x48, 0xe7, 0x4c, 0xbb,
0x92, 0x1d, 0x83, // NOLINT(readability-magic-numbers)
    0x26, 0x63, 0x16, 0x31, 0xfc, 0x6a, 0x5d, 0x30, 0x4f, 0x39, 0x16, 0x6d, 0xaf,
0x7e, 0xa1, 0x5f, // NOLINT(readability-magic-numbers)
    0xa1, 0x97, 0x7f, 0x10, 0x18, 0x19, 0xad, 0xb5, 0x10, 0xb5, 0x0f, 0xe9, 0x93,
0x2e, 0x12, 0xc5, // NOLINT(readability-magic-numbers)
    0xa8, 0x5a, 0xa3, 0xfd, 0x1e, 0x73, 0xd8, 0xd7, 0x60, 0xaf, 0x21, 0x8b, 0xe8,
0x29, 0x90, 0x3a, // NOLINT(readability-magic-numbers)
    0x77, 0xc6, 0x33, 0x59, 0xd7, 0x5e, 0xdd, 0x91, 0xb4, 0xf6, 0xed, 0x54, 0x65,
0xa7, 0x26, 0x62, // NOLINT(readability-magic-numbers)
    0xf5, 0x05, 0x59, 0x99, 0xe0, 0x59, 0xe7, 0x65, 0x4a, 0x8e, 0xdc, 0x92, 0x1a,
0xa0, 0xd4, 0x96 // NOLINT(readability-magic-numbers)
};
uint8_t PT[PT_LEN_IN_BYTE] =
{
    0xfe, 0xf0, 0x3c, 0x2d, 0x7f, 0xb1, 0x5b, 0xf0, 0xd2, 0xdf, 0x18, 0x00, 0x7d,
0x99, 0xf9, 0x67, // NOLINT(readability-magic-numbers)
    0xc8, 0x78, 0xad, 0x59, 0x35, 0x90, 0x34, 0xf7, 0xbb, 0x2c, 0x19, 0xaf, 0x12,
0x06, 0x85, 0xd7, // NOLINT(readability-magic-numbers)
    0x8e, 0x32, 0xf6, 0xb8, 0xb8, 0x3b, 0x03, 0x20, 0x19, 0x95, 0x6c, 0xa9, 0xc0,
0x19, 0x57, 0x21, // NOLINT(readability-magic-numbers)
```

```
    0x47, 0x6b,
0x85                                                                    //
NOLINT(readability-magic-numbers)
};
uint8_t ADD[AAD_LEN_IN_BYTE] =
{
    0xd8, 0xf1, 0x16, 0x3d, 0x8c, 0x84, 0x02, 0x92, 0xa2, 0xb2, 0xda, 0xcf, 0x4a,
0xc7, 0xc3, 0x6a, // NOLINT(readability-magic-numbers)
    0xff, 0x87, 0x33, 0xf1, 0x8f, 0xab, 0xb4, 0xfa, 0x55, 0x94, 0x54, 0x41, 0x25,
0xe0, 0x3d, 0x1e, // NOLINT(readability-magic-numbers)
    0x6e, 0x5d, 0x6d, 0x0f, 0xd6, 0x16, 0x56, 0xc8, 0xd8, 0xf3, 0x27, 0xc9, 0x28,
0x39, 0xae, 0x55, // NOLINT(readability-magic-numbers)
    0x39, 0xbb, 0x46, 0x9c, 0x92, 0x57, 0xf1, 0x09, 0xeb, 0xff, 0x85, 0xaa, 0xd7,
0xbd, 0x22, 0x0f, // NOLINT(readability-magic-numbers)
    0xda, 0xa9, 0x5c, 0x02, 0x2d, 0xbd, 0x0c, 0x7b, 0xb2, 0xd8, 0x78, 0xad, 0x50,
0x41, 0x22, 0xc9, // NOLINT(readability-magic-numbers)
    0x43, 0x04, 0x5d, 0x3c, 0x5e, 0xba, 0x8f, 0x1f, 0x56,
0xc0                                    // NOLINT(readability-magic-numbers)
};
uint8_t CT[PT_LEN_IN_BYTE] =
{
    0x4f, 0x6c, 0xf4, 0x71, 0xbe, 0x7c, 0xbd, 0x25, 0x75, 0xcd, 0x5a, 0x17, 0x47,
0xae, 0xa8, 0xfe, // NOLINT(readability-magic-numbers)
    0x9d, 0xea, 0x83, 0xe5, 0x19, 0x36, 0xbe, 0xac, 0x3e, 0x68, 0xf6, 0x62, 0x06,
0x92, 0x20, 0x60, // NOLINT(readability-magic-numbers)
    0xc6, 0x97, 0xff, 0xa7, 0xaf, 0x80, 0xad, 0x6b, 0xb6, 0x8f, 0x2c, 0xf4, 0xfc,
0x97, 0x41, 0x6e, // NOLINT(readability-magic-numbers)
    0xe5, 0x2a,
0xbe                                                                    //
NOLINT(readability-magic-numbers)
};
uint8_t TAG[TAG_LEN_IN_BYTE] =
{
    0xe2, 0x0b, 0x66, 0x55            // NOLINT(readability-magic-numbers)
```

```
};
void tinycrypt_aes128gcm_example (void)
{
    uint8_t gcm_encrypted[PT_LEN_IN_BYTE]      = {0};
    uint8_t gcm_tag_encrypted[TAG_LEN_IN_BYTE] = {0};
    uint8_t gcm_decrypted[PT_LEN_IN_BYTE]      = {0};
    uint16_t tlen = TAG_LEN_IN_BYTE;
 struct tc_gcm_mode_struct      g;
 struct tc_aes_key_sched_struct sched;
/* Encryption Test */
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&sched, gcm_key_128))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_gcm_config(&g, &sched, (uint8_t) tlen))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
            tc_gcm_generation_encryption(&gcm_encrypted[0], sizeof(PT),
&gcm_tag_encrypted[0],
                                         (const uint8_t *) &ADD[0], sizeof(ADD),
(const uint8_t *) &IV[0], sizeof(IV),
                                         &PT[0], sizeof(PT), &g))
    {
       debugger_break();
    }
 else if (0 != memcmp(&CT[0], &gcm_encrypted[0], sizeof(CT)))
    {
       debugger_break();
    }
 else if (0 != memcmp(&TAG[0], &gcm_tag_encrypted[0], sizeof(TAG)))
    {
       debugger_break();
```

```
    }
 else
    {
 /* Operation successful. */
    }
/* Decryption Test */
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_decrypt_key(&sched, gcm_key_128))
    {
       debugger_break();;
    }
 else if (TC_CRYPTO_SUCCESS != tc_gcm_config(&g, &sched, (uint8_t) tlen))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_gcm_decryption_verification(&gcm_decrypted[0], sizeof(CT), &TAG[0],
(const uint8_t *) &ADD[0],
 sizeof(ADD), (const uint8_t *) &IV[0], sizeof(IV), &CT[0], sizeof(CT), &g))
    {
       debugger_break();
    }
 else if (0 != memcmp(&PT[0], &gcm_decrypted[0], sizeof(PT)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
    }
}
```

### Function Documentation

### ◆ RM_TINCYRYPT_PORT_Init()

| fsp_err_t RM_TINCYRYPT_PORT_Init ( ) |
|---|
| Initialize the SCE. |

### ◆ RM_TINCYRYPT_PORT_TRNG_Read()

| fsp_err_t RM_TINCYRYPT_PORT_TRNG_Read ( uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes* ) |
|---|
| Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Random number generation successful |
| FSP_ERR_ASSERTION | NULL input parameter(s). |
| FSP_ERR_CRYPTO_UNKNOWN | An unknown error occurred. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ default_CSPRNG()

| int default_CSPRNG ( uint8_t * *dest*, unsigned int *size* ) |
|---|
| Implements the Cryptographically Secure Pseudo-Random Number Generator function required byt TinyCrypt. |

**Return values**

| | |
|---|---|
| TC_CRYPTO_SUCCESS | Random number generation successful |
| TC_CRYPTO_FAIL | Random number generation failed. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- RM_TINCYRYPT_PORT_TRNG_Read

## 5.2.16 Sensor

Modules

### Detailed Description

Sensor Modules.

### Modules

| | |
|---|---|
| | FS3000 Flow Sensor (rm_fs3000) [Deprecated]<br><br>Middleware to implement the FS3000 sensor interface. This module implements the FSXXXX Middleware Interface. |
| | HS300X Temperature/Humidity Sensor (rm_hs300x) [Deprecated]<br><br>Middleware to implement the HS300X sensor interface. This module implements the HS300X Middleware Interface. |
| | HS400X Temperature/Humidity Sensor (rm_hs400x) [Deprecated]<br><br>Middleware to implement the HS400X sensor interface. This module implements the HS400X Middleware Interface. |
| | RRH46410 Gas Sensor Module (rm_rrh46410) [Deprecated]<br><br>Middleware to implement the RRH46410 sensor module interface. This module implements the ZMOD4XXX Middleware Interface. |
| | RRH47000 NDIR CO2 Sensor Module (rm_rrh47000) [Deprecated]<br><br>Middleware to implement the RRH47000 sensor module interface. This module implements the SENSOR Middleware Interface. |
| | RRH62000 All-in-one Air Quality Module (rm_rrh62000) [Deprecated]<br><br>Middleware to implement the RRH62000 sensor module interface. This module implements the SENSOR Middleware Interface. |
| | ZMOD4XXX Gas Sensor (rm_zmod4xxx) [Deprecated]<br><br>Middleware to implement the ZMOD4XXX sensor interface. This module implements the ZMOD4XXX Middleware Interface. |

## 5.2.16.1 FS3000 Flow Sensor (rm_fs3000) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_FS3000_Open (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_cfg_t const *const p_cfg)<br><br>Opens and configures the FS3000 Middle module. Implements |

| | |
|---|---|
| | rm_fsxxxx_api_t::open. More... |
| fsp_err_t | RM_FS3000_Close (rm_fsxxxx_ctrl_t *const p_api_ctrl)<br>Disables specified FS3000 control block. Implements rm_fsxxxx_api_t::close. More... |
| fsp_err_t | RM_FS3000_Read (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data)<br>Reads ADC data from FS3000. Implements rm_fsxxxx_api_t::read. More... |
| fsp_err_t | RM_FS3000_DataCalculate (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fs3000_data)<br>Calculates flow [m/sec] from ADC data. Implements rm_fsxxxx_api_t::dataCalculate. More... |

### Detailed Description

Middleware to implement the FS3000 sensor interface. This module implements the FSXXXX Middleware Interface.

# Overview

FS3000 is EOL.

### Features

The FS3000 sensor interface implementation has the following key features:

- Getting ADC data from the sensor
- Calculating flow value from ADC data

### Limitations

- I2C Master (r_sau_i2c) is not supported. FS3000 needs clock stretching, but is not supported by SAU I2C.

# Configuration

### Build Time Configurations for rm_fs3000

The following build time configurations are defined in fsp_cfg/rm_fs3000_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Device Type | FS3000-1005 | FS3000-1005 | Select FS3000 device used. |

**Configurations for Sensor > FS3000 Flow Sensor (rm_fs3000) [Deprecated]**

This module can be added to the Stacks tab via New Stack > Sensor > FS3000 Flow Sensor (rm_fs3000) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_fs3000_sensor0 | Module name. |
| Callback | Name must be a valid C symbol | fs3000_callback | A user callback function can be provided. |

**Pin Configuration**

This module use SDA and SCL pins of I2C Master and SCI I2C.

# Usage Notes

FS3000 datasheet is here.
If ADC data is invalid, it is needed to read ADC data from FS3000 again. The module only supports FS3000-1005.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

**Bus Initialization**

The FS3000 interface expects a bus instance to be opened before opening any FS3000 device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

# Examples

**Basic Example**

This is a basic example of minimal use of FS3000 sensor implementation in an application.

```
void rm_fs3000_basic_example (void)

{
 fsp_err_t            err = FSP_SUCCESS;
```

```
 rm_fsxxxx_raw_data_t fs3000_raw_data;
 rm_fsxxxx_data_t    fs3000_data;
    uint8_t             calculated_flag = 0;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *) g_fs3000_cfg.p_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
```

```
      *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
           xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_FS3000_Open(&g_fs3000_ctrl, &g_fs3000_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
 do
     {
           g_flag = 0;
 /* Read ADC Data from FS3000 */
 RM_FS3000_Read(&g_fs3000_ctrl, &fs3000_raw_data);
 while (0 == g_flag)
     {
 /* Wait callback */
     }
 /* Calculate Flow value from ADC data */
           err = RM_FS3000_DataCalculate(&g_fs3000_ctrl, &fs3000_raw_data,
&fs3000_data);
 if (FSP_SUCCESS == err)
     {
               calculated_flag = 1;
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err) /* Checksum error */
     {
               calculated_flag = 0;
     }
 else
     {
     handle_error(err);
```

```
      }
      } while (0 == calculated_flag);
 /* Wait 125 milliseconds. See table 4 on the page 7 of the datasheet. */
 R_BSP_SoftwareDelay(FS3000_DELAY_125, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_fs3000_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_fs3000_instance_ctrl_t

| struct rm_fs3000_instance_ctrl_t | |
|---|---|
| FS3000 Control Block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_fsxxxx_cfg_t const * | p_cfg |
| | Pointer to FS3000 Configuration. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |

## Function Documentation

### ◆ RM_FS3000_Open()

fsp_err_t RM_FS3000_Open ( rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_cfg_t const *const *p_cfg* )

Opens and configures the FS3000 Middle module. Implements rm_fsxxxx_api_t::open.

Example:

```
    err = RM_FS3000_Open(&g_fs3000_ctrl, &g_fs3000_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | FS3000 successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ RM_FS3000_Close()

fsp_err_t RM_FS3000_Close ( rm_fsxxxx_ctrl_t *const *p_api_ctrl*)

Disables specified FS3000 control block. Implements rm_fsxxxx_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_FS3000_Read()

fsp_err_t RM_FS3000_Read ( rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_raw_data_t *const *p_raw_data* )

Reads ADC data from FS3000. Implements rm_fsxxxx_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_FS3000_DataCalculate()

| fsp_err_t RM_FS3000_DataCalculate ( rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_raw_data_t *const *p_raw_data*, rm_fsxxxx_data_t *const *p_fs3000_data* ) |
|---|
| Calculates flow [m/sec] from ADC data. Implements rm_fsxxxx_api_t::dataCalculate. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_SENSOR_INVALID_DATA | Data is invalid. |

## 5.2.16.2 HS300X Temperature/Humidity Sensor (rm_hs300x) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_HS300X_Open (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_cfg_t const *const p_cfg)<br>Opens and configures the HS300X Middle module. Implements rm_hs300x_api_t::open. More... |
| fsp_err_t | RM_HS300X_Close (rm_hs300x_ctrl_t *const p_api_ctrl)<br>Disables specified HS300X control block. Implements rm_hs300x_api_t::close. More... |
| fsp_err_t | RM_HS300X_MeasurementStart (rm_hs300x_ctrl_t *const p_api_ctrl)<br>This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the hs300x and start a measurement. Implements rm_hs300x_api_t::measurementStart. More... |
| fsp_err_t | RM_HS300X_Read (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_raw_data_t *const p_raw_data)<br>Reads ADC data from HS300X. Implements rm_hs300x_api_t::read. More... |

| | |
|---|---|
| fsp_err_t | RM_HS300X_DataCalculate (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data)<br>Calculates humidity [RH] and temperature [Celsius] from ADC data. Implements rm_hs300x_api_t::dataCalculate. More... |
| fsp_err_t | RM_HS300X_ProgrammingModeEnter (rm_hs300x_ctrl_t *const p_api_ctrl)<br>This function must be called within 10ms after applying power to the sensor. Sends the commands to enter the programming mode. After calling this function, please wait 120us. Implements rm_hs300x_api_t::programmingModeEnter. More... |
| fsp_err_t | RM_HS300X_ResolutionChange (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution)<br>This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Changes the sensor resolution. This function blocks for 120 us software delay plus 9 bytes on the I2C bus. After calling this function, 14ms must be waited. Failure to comply with these times may result in data corruption and introduce errors in sensor measurements. Implements rm_hs300x_api_t::resolutionChange. More... |
| fsp_err_t | RM_HS300X_SensorIdGet (rm_hs300x_ctrl_t *const p_api_ctrl, uint32_t *const p_sensor_id)<br>This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Gets the sensor ID. This function blocks for 240 us software delay plus 12 bytes on the I2C bus. Implements rm_hs300x_api_t::sensorIdGet. More... |
| fsp_err_t | RM_HS300X_ProgrammingModeExit (rm_hs300x_ctrl_t *const p_api_ctrl)<br>This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. This function must be called to return to normal sensor operation and perform measurements. Sends the commands to exit the programming mode. Implements rm_hs300x_api_t::programmingModeExit. More... |

## Detailed Description

Middleware to implement the HS300X sensor interface. This module implements the HS300X Middleware Interface.

# Overview

HS300x is EOL.

### Features

The HS300X sensor interface implementation has the following key features:

- Starting a measurement at any time
- Getting ADC data from the sensor
- Calculating humidity and temperature value from getting ADC data
- Changing the sensor resolution
- Getting the sensor ID

# Configuration

### Build Time Configurations for rm_hs300x

The following build time configurations are defined in fsp_cfg/rm_hs300x_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Data type | • Both humidity and temperature<br>• Humidity only | Both humidity and temperature | Select Getting humidity only and both humidity and temperature. |
| Programming Mode | • ON<br>• OFF | OFF | If selected the programming mode can be entered. |

### Configurations for Sensor > HS300X Temperature/Humidity Sensor (rm_hs300x) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > HS300X Temperature/Humidity Sensor (rm_hs300x) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hs300x_sensor0 | Module name. |
| Callback | Name must be a valid C symbol | hs300x_callback | A user callback function can be provided. |

### Pin Configuration

This module use SDA and SCL pins of I2C Master and SCI I2C.

# Usage Notes

HS300x datasheet is here.
If ADC data is valid and calculating humidity and temperature is finished, it is needed to start a measurement again. If ADC data is invalid, it is needed to read ADC data from HS300x again.

If changing the sensor resolution and getting the sensor ID, RM_HS300X_ProgrammingModeEnter function must be called within 10ms after applying power to the sensor. Entering the programming mode takes 120us. Thresore, after calling RM_HS300X_ProgrammingModeEnter function, please wait 120us. After calling RM_HS300X_ResolutionChange function, 14ms must be waited because failure to comply with these times may result in data corruption and introduce errors in sensor measurements.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

### Bus Initialization

The HS300X interface expects a bus instance to be opened before opening any HS300X device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

## Examples

### Basic Example

This is a basic example of minimal use of HS300X sensor implementation in an application.

```
void rm_hs300x_basic_example (void)
{
 fsp_err_t            err = FSP_SUCCESS;
 rm_hs300x_raw_data_t hs300x_raw_data;
 rm_hs300x_data_t    hs300x_data;
    uint8_t             calculated_flag = 0;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *) g_hs300x_cfg.p_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
```

```
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_HS300X_Open(&g_hs300x_ctrl, &g_hs300x_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if RM_HS300X_CFG_PROGRAMMING_MODE
    uint32_t sensor_id;
    g_flag = 0;
 /* Enter the programming mode. This must be called within 10ms after applying power.
```

```
*/
 RM_HS300X_ProgrammingModeEnter(&g_hs300x_ctrl);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 120us. Entering the programming mode takes 120us. */
 R_BSP_SoftwareDelay(120, BSP_DELAY_UNITS_MICROSECONDS);
 /* Get the sensor ID */
 RM_HS300X_SensorIdGet(&g_hs300x_ctrl, &sensor_id);
    g_flag = 0;
 /* Change the humidity resolution */
 RM_HS300X_ResolutionChange(&g_hs300x_ctrl, RM_HS300X_HUMIDITY_DATA,
RM_HS300X_RESOLUTION_8BIT);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 14ms. Failure to comply with these times may result in data corruption and
introduce errors in sensor measurements. */
 R_BSP_SoftwareDelay(14, BSP_DELAY_UNITS_MILLISECONDS);
    g_flag = 0;
 /* Change the temperature resolution */
 RM_HS300X_ResolutionChange(&g_hs300x_ctrl, RM_HS300X_TEMPERATURE_DATA,
RM_HS300X_RESOLUTION_8BIT);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 14ms. Failure to comply with these times may result in data corruption and
introduce errors in sensor measurements. */
 R_BSP_SoftwareDelay(14, BSP_DELAY_UNITS_MILLISECONDS);
    g_flag = 0;
 /* Exit the programming mode */
```

```
 RM_HS300X_ProgrammingModeExit(&g_hs300x_ctrl);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
#endif
 while (true)
    {
      g_flag = 0;
 /* Start Measurement */
 RM_HS300X_MeasurementStart(&g_hs300x_ctrl);
 while (0 == g_flag)
     {
 /* Wait callback */
     }
 do
     {
     g_flag = 0;
 /* Read ADC Data from HS300X */
 RM_HS300X_Read(&g_hs300x_ctrl, &hs300x_raw_data);
 while (0 == g_flag)
     {
 /* Wait callback */
     }
 /* Calculate Humidity and Temperatuere values from ADC data */
           err = RM_HS300X_DataCalculate(&g_hs300x_ctrl, &hs300x_raw_data,
&hs300x_data);
 if (FSP_SUCCESS == err)
     {
               calculated_flag = 1;
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Stale data */
```

```
                calculated_flag = 0;
      }
 else
      {
      handle_error(err);
      }
      } while (0 == calculated_flag);
 /* Wait 4 seconds. See table 4 on the page 6 of the datasheet. */
 R_BSP_SoftwareDelay(4, BSP_DELAY_UNITS_SECONDS);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_hs300x_programmnig_mode_params_t |
| struct | rm_hs300x_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_hs300x_programmnig_mode_params_t

| struct rm_hs300x_programmnig_mode_params_t | | |
|---|---|---|
| HS300X programming mode process block | | |
| Data Fields | | |
| volatile bool | enter | Enter flag. |
| volatile bool | blocking | Blocking flag. |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile rm_hs300x_event_t | event | Callback event. |

### ◆ rm_hs300x_instance_ctrl_t

| struct rm_hs300x_instance_ctrl_t | |
|---|---|
| HS300x Control Block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| | |

| | |
|---|---|
| rm_hs300x_cfg_t const * | p_cfg |
| | Pointer to HS300X Configuration. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| rm_hs300x_programmnig_mode_params_t | programming_mode |
| | Programming mode flag. |
| | |
| uint8_t | buf [3] |
| | Buffer for I2C communications. |
| | |

### Function Documentation

#### ◆ RM_HS300X_Open()

fsp_err_t RM_HS300X_Open ( rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_cfg_t const *const *p_cfg* )

Opens and configures the HS300X Middle module. Implements rm_hs300x_api_t::open.

Example:

```
    err = RM_HS300X_Open(&g_hs300x_ctrl, &g_hs300x_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | HS300X successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |

### ◆ RM_HS300X_Close()

fsp_err_t RM_HS300X_Close ( rm_hs300x_ctrl_t *const *p_api_ctrl*)

Disables specified HS300X control block. Implements rm_hs300x_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_HS300X_MeasurementStart()

fsp_err_t RM_HS300X_MeasurementStart ( rm_hs300x_ctrl_t *const *p_api_ctrl*)

This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the hs300x and start a measurement. Implements rm_hs300x_api_t::measurementStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_HS300X_Read()

fsp_err_t RM_HS300X_Read ( rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_raw_data_t *const *p_raw_data* )

Reads ADC data from HS300X. Implements rm_hs300x_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_HS300X_DataCalculate()

fsp_err_t RM_HS300X_DataCalculate ( rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_raw_data_t *const *p_raw_data*, rm_hs300x_data_t *const *p_hs300x_data* )

Calculates humidity [RH] and temperature [Celsius] from ADC data. Implements rm_hs300x_api_t::dataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_SENSOR_INVALID_DATA | Data is invalid. |

### ◆ RM_HS300X_ProgrammingModeEnter()

fsp_err_t RM_HS300X_ProgrammingModeEnter ( rm_hs300x_ctrl_t *const *p_api_ctrl*)

This function must be called within 10ms after applying power to the sensor. Sends the commands to enter the programming mode. After calling this function, please wait 120us. Implements rm_hs300x_api_t::programmingModeEnter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_UNSUPPORTED | Programming mode is not supported. |

### ◆ RM_HS300X_ResolutionChange()

fsp_err_t RM_HS300X_ResolutionChange ( rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_data_type_t const *data_type*, rm_hs300x_resolution_t const *resolution* )

This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Changes the sensor resolution. This function blocks for 120 us software delay plus 9 bytes on the I2C bus. After calling this function, 14ms must be waited. Failure to comply with these times may result in data corruption and introduce errors in sensor measurements. Implements rm_hs300x_api_t::resolutionChange.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Module is not entering the programming mode. |
| FSP_ERR_ABORTED | Communication is aborted. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_UNSUPPORTED | Programming mode is not supported. |

### ◆ RM_HS300X_SensorIdGet()

fsp_err_t RM_HS300X_SensorIdGet ( rm_hs300x_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_sensor_id* )

This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Gets the sensor ID. This function blocks for 240 us software delay plus 12 bytes on the I2C bus. Implements rm_hs300x_api_t::sensorIdGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Module is not entering the programming mode. |
| FSP_ERR_ABORTED | Communication is aborted. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_UNSUPPORTED | Programming mode is not supported. |

### ◆ RM_HS300X_ProgrammingModeExit()

| fsp_err_t RM_HS300X_ProgrammingModeExit ( rm_hs300x_ctrl_t *const *p_api_ctrl*) |
|---|
| This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. This function must be called to return to normal sensor operation and perform measurements. Sends the commands to exit the programming mode. Implements rm_hs300x_api_t::programmingModeExit. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_INVALID_MODE | Module is not entering the programming mode. |
| FSP_ERR_UNSUPPORTED | Programming mode is not supported. |

## 5.2.16.3 HS400X Temperature/Humidity Sensor (rm_hs400x) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_HS400X_Open (rm_hs400x_ctrl_t *const p_api_ctrl, rm_hs400x_cfg_t const *const p_cfg)<br>Opens and configures the HS400X middleware module. Implements rm_hs400x_api_t::open. More... |
| fsp_err_t | RM_HS400X_Close (rm_hs400x_ctrl_t *const p_api_ctrl)<br>Disables specified HS400X control block. Implements rm_hs400x_api_t::close. More... |
| fsp_err_t | RM_HS400X_MeasurementStart (rm_hs400x_ctrl_t *const p_api_ctrl)<br>This function should be called when start one shot measurement. Sends the command of measurement to HS400X and start a measurement. This function supports No-Hold measurement and Periodic measurement only. If Hold measurement is enabled, please call RM_HS400X_Read() without calling this function. In Periodic measurement, if the periodic measurement has already run, RM_HS400X_EVENT_ERROR is received in callback because HS400x device replies with NACK. Implements rm_hs400x_api_t::measurementStart. More... |

| | |
|---|---|
| fsp_err_t | RM_HS400X_MeasurementStop (rm_hs400x_ctrl_t *const p_api_ctrl)<br>Stop a periodic measurement. Sends the command of stopping periodic measurement to HS400X. This function supports periodic measurement only. If a periodic measurement is not running, RM_HS400X_EVENT_ERROR is received in callback because HS400x device replies with NACK. Implements rm_hs400x_api_t::measurementStop. More... |
| fsp_err_t | RM_HS400X_Read (rm_hs400x_ctrl_t *const p_api_ctrl, rm_hs400x_raw_data_t *const p_raw_data)<br>Reads ADC data from HS400X. If Hold measurement is enabled, HS400X holds the SCL line low during the measurement and releases the SCL line when the measurement is complete. If No-Hold measurement is enabled and the measurement result is not ready, RM_HS400X_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback. Implements rm_hs400x_api_t::read. More... |
| fsp_err_t | RM_HS400X_DataCalculate (rm_hs400x_ctrl_t *const p_api_ctrl, rm_hs400x_raw_data_t *const p_raw_data, rm_hs400x_data_t *const p_hs400x_data)<br>Calculates temperature [Celsius] and humidity [RH] from ADC data. Implements rm_hs400x_api_t::dataCalculate. More... |

### Detailed Description

Middleware to implement the HS400X sensor interface. This module implements the HS400X Middleware Interface.

# Overview

HS400x is EOL.

### Features

The HS400X sensor interface implementation has the following key features:

- Starting a measurement at any time
- Selecting one shot or a periodic measurement
- Getting ADC data from the sensor
- Calculating humidity and temperature value from getting ADC data

### Limitations

- Alert feature is not supported. It will be supported in the future.
- I2C Master (r_sau_i2c) is not supported in Hold measurement. Hold measurement needs clock stretching, but is not supported by SAU I2C.

### Notifications

The minimum frequency for the SCL clock in Hold measurement is 200kHz. Please configure I2C_MASTER_RATE_FAST or I2C_MASTER_RATE_FASTPLUS to "Rate" in I2C Master (r_iic_master) or I2C Master (r_sci_i2c).

# Configuration

### Build Time Configurations for rm_hs400x

The following build time configurations are defined in fsp_cfg/rm_hs400x_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Measurement Type | • Hold Measurement<br>• No-Hold Measurement<br>• Periodic Measurement | No-Hold Measurement | Select measurement type. |
| Data type | • Both humidity and temperature<br>• Temperature only | Both humidity and temperature | Select getting temperature only or both humidity and temperature. |

### Configurations for Sensor > HS400X Temperature/Humidity Sensor (rm_hs400x) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > HS400X Temperature/Humidity Sensor (rm_hs400x) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_hs400x_sensor0 | Module name. |
| Temperature Resolution | • 8-bit<br>• 10-bit<br>• 12-bit<br>• 14-bit | 14-bit | Set resolution of temperature. |
| Humidity Resolution | • 8-bit<br>• 10-bit<br>• 12-bit<br>• 14-bit | 14-bit | Set resolution of humidity. |
| Frequency for Periodic Measurement | • 0.4Hz<br>• 1Hz<br>• 2Hz | 1Hz | Set frequency for periodic measurement. |
| Comms I2C Callback | Name must be a valid | hs400x_comms_i2c_cal | A user COMMS I2C |

| | | | |
|---|---|---|---|
| | C symbol | lback | callback function can be provided. |

### Pin Configuration

This module use SDA and SCL pins of I2C Master and SCI I2C.

# Usage Notes

HS400x datasheet is here.
This module supports three measurement type :

- Hold measurement
- No-Hold measurement
- Periodic measurement

Hold and no-hold measurements also support measuring temperature data only. However, a periodic measurement only supports measuring both humidity and temperature data.

### Hold Measurement

A hold measurement sequence consists of the following steps.

1. Wake up the HS400x series sensor from sleep mode by sending its I2C address with a write bit, and initiate a measurement by sending the desired hold measurement command.
2. Change the direction of communication by sending a start bit, the HS400x I2C address, and a read bit. The SCL line is held low by the sensor during the measurement process, which prevents the master from initiating any communications with other slaves on the bus.
3. Once the requested measurement is completed by the HS400x series sensor, the SCL line is released and the chip waits for the SCL clock signal to send the results. The sensor will then transmit the requested measurement data on the bus for the master to capture.

### No-Hold Measurement

A no-hold measurement sequence consists of the following steps.

1. Wake up the HS400x series sensor from sleep mode by sending its I2C address with a write bit, and initiate a measurement by sending the desired no-hold measurement command.
2. To read the result from the HS400x series sensor, the master has to send the chip its I2C address and a read bit. If the measurement is completed and the result is ready, the chip will send an ACK bit and starts to send the result over the bus. If the measurement is still in progress, the chip will send a NACK bit and the master will need to try to read the result again.

### Periodic Measurement

The HS400x sensors can also be configured to measure at regular intervals without user intervention. In this mode, the user can read the latest relative humidity / temperature data by issuing a data fetch sequence, which consists of sending the HS400x I2C address with a read bit. The sensor will then send the latest measurement result over the I2C bus.
When the periodic measurement mode is active, the only commands the chip will respond to are the data fetch command, and a command to stop the periodic measurements. The command to stop periodic measurements is issued by sending the I2C address with a write bit, followed by the command 0x30. Sequence to Stop Periodic Measurements. Once the periodic measurements have been stopped, the chip returns to sleep and is ready to accept all valid I2C commands.

### Bus Initialization

The HS400X interface expects a I2C bus instance to be opened before opening any I2C device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

### Initialization

Initialize with RM_HS400X_Open().

### From measurement start to data acquisition

After normal completion, start the each measurement.

### Hold Measurement

In hold measurement, the HS400x series sensor holds the SCL line low during the measurement and releases the SCL line when the measurement is complete.

1. Call RM_HS400X_Read(). This function will start a measurement and read the ADC data.
2. Wait until RM_HS400X_EVENT_SUCCESS is received.
3. Call RM_HS400X_DataCalculate(). This function will calculate temperature and humidity data from the ADC data.

### No-Hold Measurement

In no-hold measurement, the HS400x series sensor does not hold the SCL line low, and the master is free to initiate communication with other slaves while the chip is performing the measurement.

1. Call RM_HS400X_MeasurementStart(). This function will start a measurement.
2. Wait until RM_HS400X_EVENT_SUCCESS is received.
3. Call RM_HS400X_Read(). This function will read the ADC data.
4. If the measurement result is not ready, RM_HS400X_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback. User should call RM_HS400X_Read() again. If the measurement is completed, RM_HS400X_EVENT_SUCCESS is received in callback.
5. Wait until RM_HS400X_EVENT_SUCCESS is received.
6. Call RM_HS400X_DataCalculate(). This function will calculate temperature and humidity data from the ADC data.

### Periodic Measurement

In periodic measurement, the HS400x sensors measure at regular intervals without user intervention. If a periodic measurement is running, the resolution and the frequency cannot be changed. please stop the periodic measurement with RM_HS400X_MeasurementStop().

1. Call RM_HS400X_MeasurementStart(). This function will start a measurement. If the function is called once, a second call is not required.
2. Wait until RM_HS400X_EVENT_SUCCESS is received.

3. Wait for frequency for periodic measurement.
4. Call RM_HS400X_Read(). This function will read the ADC data.
5. Wait until RM_HS400X_EVENT_SUCCESS is received.
6. Call RM_HS400X_DataCalculate(). This function will calculate temperature and humidity data from the ADC data.

# Examples

### Basic Example

This is a basic examples of minimal use of HS400X sensor implementation in an application.

```
void rm_hs400x_basic_example (void)
{
 fsp_err_t            err = FSP_SUCCESS;
 rm_hs400x_raw_data_t hs400x_raw_data;
 rm_hs400x_data_t     hs400x_data;
 bool                 measurement_complete = false;
 bool                 calculated           = false;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_hs400x_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
```

```
                                        (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_HS400X_Open(&g_hs400x_ctrl, &g_hs400x_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if 3 == RM_HS400X_CFG_MEASUREMENT_TYPE // Periodic Measurement
    g_flag = 0;
 /* Start a periodic measurement */
 RM_HS400X_MeasurementStart(&g_hs400x_ctrl);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
#endif
 while (true)
    {
#if 2 == RM_HS400X_CFG_MEASUREMENT_TYPE // No-Hold Measurement
```

```
     g_flag = 0;
 /* Start one shot measurement */
 RM_HS400X_MeasurementStart(&g_hs400x_ctrl);
 while (0 == g_flag)
     {
 /* Wait callback */
     }
#elif 3 == RM_HS400X_CFG_MEASUREMENT_TYPE // Periodic Measurement
 /* Wait Periodic measurement period. See table 12 on the page 20 of the datasheet.
*/
 switch (g_hs400x_cfg.frequency)
     {
 case RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_2HZ:
 R_BSP_SoftwareDelay(500, BSP_DELAY_UNITS_MILLISECONDS);
 break;
 case RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_1HZ:
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
 break;
 case RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_0P4HZ:
 R_BSP_SoftwareDelay(2500, BSP_DELAY_UNITS_MILLISECONDS);
 break;
 default:
               assert(false);
 break;
     }
#else
#endif
 do
     {
 do
     {
     g_flag = 0;
 /* Read ADC Data from HS400X */
 RM_HS400X_Read(&g_hs400x_ctrl, &hs400x_raw_data);
```

```
 while (0 == g_flag)
      {
 /* Wait callback */
      }
#if 2 == RM_HS400X_CFG_MEASUREMENT_TYPE // No-Hold Measurement
 if (RM_HS400X_EVENT_MEASUREMENT_NOT_COMPLETE == g_hs400x_event)
      {
 /* RM_HS400X_EVENT_MEASUREMENT_NOT_COMPLETE is received */
                    measurement_complete = false;
      }
 else
      {
 /* RM_HS400X_EVENT_SUCCESS is received. */
                    measurement_complete = true;
      }
#else
                measurement_complete = true;
#endif
      } while (false == measurement_complete);
 /* Calculate Humidity and Temperature values from ADC data */
            err = RM_HS400X_DataCalculate(&g_hs400x_ctrl, &hs400x_raw_data,
&hs400x_data);
 if (FSP_SUCCESS == err)
      {
                calculated = true;
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* CRC error */
                calculated = false;
      }
 else
      {
      handle_error(err);
```

```
      }
      } while (false == calculated);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_hs400x_init_process_params_t |
| struct | rm_hs400x_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_hs400x_init_process_params_t

| struct rm_hs400x_init_process_params_t | | |
|---|---|---|
| HS400X initialization process block | | |
| Data Fields | | |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile rm_hs400x_event_t | event | Callback event. |

### ◆ rm_hs400x_instance_ctrl_t

| struct rm_hs400x_instance_ctrl_t | |
|---|---|
| HS400x Control Block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| rm_hs400x_cfg_t const * | p_cfg |
| | Pointer to HS400X Configuration. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void * | p_context |
| | |

| | |
|---|---|
| | Pointer to the user-provided context. |
| | |
| rm_hs400x_init_process_params_t | init_process_params |
| | For the initialization process. |
| | |
| uint8_t | resolution_register |
| | Register for temperature and humidity measurement resolution settings. |
| | |
| uint8_t | periodic_measurement_register [2] |
| | Register for periodic measurement settings. |
| | |
| volatile bool | periodic_measurement_stop |
| | Flag for stop of periodic measurement. |
| | |
| volatile bool | no_hold_measurement_read |
| | Flag for data read of No-Hold measurement. |
| | |
| uint8_t | write_buf [18] |
| | Buffer for data write. |
| | |
| void(* | p_comms_callback )(rm_hs400x_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |

### Function Documentation

### ◆ RM_HS400X_Open()

fsp_err_t RM_HS400X_Open ( rm_hs400x_ctrl_t *const *p_api_ctrl*, rm_hs400x_cfg_t const *const *p_cfg* )

Opens and configures the HS400X middleware module. Implements rm_hs400x_api_t::open.

Example:

```
err = RM_HS400X_Open(&g_hs400x_ctrl, &g_hs400x_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | HS400X successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_HS400X_Close()

fsp_err_t RM_HS400X_Close ( rm_hs400x_ctrl_t *const *p_api_ctrl*)

Disables specified HS400X control block. Implements rm_hs400x_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_HS400X_MeasurementStart()

fsp_err_t RM_HS400X_MeasurementStart ( rm_hs400x_ctrl_t *const *p_api_ctrl*)

This function should be called when start one shot measurement. Sends the command of measurement to HS400X and start a measurement. This function supports No-Hold measurement and Periodic measurement only. If Hold measurement is enabled, please call RM_HS400X_Read() without calling this function. In Periodic measurement, if the periodic measurement has already run, RM_HS400X_EVENT_ERROR is received in callback because HS400x device replies with NACK. Implements rm_hs400x_api_t::measurementStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |
| FSP_ERR_UNSUPPORTED | Hold measurement is unsupported. |

### ◆ RM_HS400X_MeasurementStop()

fsp_err_t RM_HS400X_MeasurementStop ( rm_hs400x_ctrl_t *const *p_api_ctrl*)

Stop a periodic measurement. Sends the command of stopping periodic measurement to HS400X. This function supports periodic measurement only. If a periodic measurement is not running, RM_HS400X_EVENT_ERROR is received in callback because HS400x device replies with NACK. Implements rm_hs400x_api_t::measurementStop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |
| FSP_ERR_UNSUPPORTED | Hold and No-Hold measurement are unsupported. |

### ◆ RM_HS400X_Read()

fsp_err_t RM_HS400X_Read ( rm_hs400x_ctrl_t *const *p_api_ctrl*, rm_hs400x_raw_data_t *const *p_raw_data* )

Reads ADC data from HS400X. If Hold measurement is enabled, HS400X holds the SCL line low during the measurement and releases the SCL line when the measurement is complete. If No-Hold measurement is enabled and the measurement result is not ready, RM_HS400X_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback. Implements rm_hs400x_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_HS400X_DataCalculate()

fsp_err_t RM_HS400X_DataCalculate ( rm_hs400x_ctrl_t *const *p_api_ctrl*, rm_hs400x_raw_data_t *const *p_raw_data*, rm_hs400x_data_t *const *p_hs400x_data* )

Calculates temperature [Celsius] and humidity [RH] from ADC data. Implements rm_hs400x_api_t::dataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully data decoded. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_SENSOR_INVALID_DATA | Data is invalid. |

## 5.2.16.4 RRH46410 Gas Sensor Module (rm_rrh46410) [Deprecated]

Modules » Sensor

**Functions**

| | |
|---|---|
| fsp_err_t | RM_RRH46410_Open (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg) |

| | |
|---|---|
| | Opens and configures the RRH46410 sensor module. Implements rm_zmod4xxx_api_t::open. More... |
| fsp_err_t | RM_RRH46410_Close (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when close the sensor module. Implements rm_zmod4xxx_api_t::close. More... |
| fsp_err_t | RM_RRH46410_MeasurementStart (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart. More... |
| fsp_err_t | RM_RRH46410_MeasurementStop (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop. More... |
| fsp_err_t | RM_RRH46410_Read (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data)<br>This function should be called to get measurement results after measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read. More... |
| fsp_err_t | RM_RRH46410_Iaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_rrh46410_data)<br>This function should be called when calculating gas data from measurement results. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate. More... |
| fsp_err_t | RM_RRH46410_PbaqDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_pbaq_data_t *const p_rrh46410_data)<br>This function should be called when calculating gas data from measurement results. Implements rm_zmod4xxx_api_t::pbaqDataCalculate. More... |
| fsp_err_t | RM_RRH46410_TemperatureAndHumiditySet (rm_zmod4xxx_ctrl_t *const p_api_ctrl, float temperature, float humidity)<br>This function should be called before Read. Humidity measurements |

| | |
|---|---|
| | are needed for ambient compensation. temperature is not supported. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet. More... |

### Detailed Description

Middleware to implement the RRH46410 sensor module interface. This module implements the ZMOD4XXX Middleware Interface.

# Overview

RRH46410 is EOL.

This module provides an API for configuring and controlling the RRH46410 sensor module (ZMOD4410 + MCU).

I2C communication with the RRH46410 sensor module is realized by connecting with the rm_comms_i2c module.

### Features

The RRH46410 sensor interface implementation has the following key features:

- Initialize the sensor for measurement
- Start a measurement at any time
- Input current humidity for best performance.
- Receive status for wait until the measurement is done. This will also be signaled by an interrupt
- Get the measruement results from the sensor module
- Different from ZMOD sensors, no libraries are needed.

# Configuration

### Build Time Configurations for rm_rrh46410

The following build time configurations are defined in fsp_cfg/rm_rrh46410_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Operation Mode | • IAQ 2nd Gen and Rel IAQ<br>• IAQ 2nd Gen ULP and Rel IAQ ULP<br>• Public Building AQ Standard (PBAQ) | IAQ 2nd Gen and Rel IAQ | Select operation mode. |

### Configurations for Sensor > RRH46410 Gas Sensor Module (rm_rrh46410) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > RRH46410 Gas Sensor Module (rm_rrh46410) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rrh46410_sensor0 | Module name. |
| Comms I2C Callback | Name must be a valid C symbol | rrh46410_comms_i2c_callback | A user COMMS I2C callback function can be provided. |
| IRQ Callback | Name must be a valid C symbol | rrh46410_irq_callback | A user IRQ callback function can be provided. |

#### Pin Configuration

This module use SDA and SCL pins of I2C Master, SCI I2C and IICA Master.

## Usage Notes

RRH46410 datasheet is here.
The RRH46410 has three modes of operation.
The RRH46410 will respond to TVOC immediately upon start-up; however, a conditioning period of 48 hours followed by a sensor module restart in an ambient environment is recommended to improve stability and obtain maximum performance.
Best results are achieved with continuous operation because the module algorithm can learn about the environment over time.

| Operation mode | Description | Data |
|---|---|---|
| IAQ 2nd Generation and Relative IAQ | Using AI for improved ppm TVOC, IAQ and eCO2 functionality (recommended for new designs). Lightweight algorithm reacts to air quality changes and outputs a relative IAQ | IAQ, TVOC[mg/m^3], EtOH[ppm], eCO2[ppm], Rel IAQ |
| IAQ 2nd Generation Ultra Low Power and Relative IAQ Ultra Low Power | Using AI for improved ppm TVOC, IAQ and eCO2 functionality. Lightweight algorithm reacts to air quality changes and outputs a relative IAQ. This operation mode offers a much lower power consumption. | IAQ, TVOC[mg/m^3], EtOH[ppm], eCO2[ppm], Rel IAQ |
| Public Building AQ Standard (PBAQ) | For highly accurate and consistent sensor readings to fulfill public building standards. | TVOC[mg/m^3], EtOH[ppm] |

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

### Limitations

- I2C Master (r_sau_i2c) is not supported. RRH46410 needs clock stretching, but is not supported by SAU I2C.
- The following commands are currently not supported.
    - Get Product ID
    - Get Tracking Number
    - Set Operation Mode -> Sensor cleaning
    - Config GPIO
    - Get GPIO
    - Set GPIO
    - Clear GPIO
    - Read Flash Shadow
    - Write Flash Shadow
    - Write Flash
    - Reset

### Notifications

- The RRH46410 sensor module needs a reset before operation; please input a reset signal to the RES_N pin (active low).
- The RRH46410 sensor module needs some time (e.g. 1s) for self initialization after power-on and reset.

### Bus Initialization

The RRH46410 interface expects a bus instance to be opened before opening any RRH46410 device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

### Initialization

Initialize with RM_RRH46410_Open().

### From measurement start to data acquisition

After normal completion, start the measurement with RM_RRH46410_MeasurementStart(). An endless loop continuously checks the status of the RRH46410 sensor module and reads its data. The measurement results are subsequently processed, and the air quality values are calculated.

### If IRQ is enabled

1. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received via IRQ callback.
2. Call RM_RRH46410_Read(). This function will read the measurement results.
3. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received.
4. Call the DataCalculate API according to the mode.

### If IRQ is disabled

1. Call RM_RRH46410_Read(). This function will read the measurement results.
2. If RM_ZMOD4XXX_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback, user

should wait some time and then call RM_RRH46410_Read() again.
3. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received.
4. Call the DataCalculate API according to the mode.

# Examples

### Basic Example

These are basic examples of minimal use of RRH46410 sensor implementation in an application.

IAQ 2nd Gen.

```
void rm_rrh46410_iaq_2nd_gen_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t rrh46410_data;
 float humidity = RRH46410_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_rrh46410_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,
```

```
p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset sensor module (active low). Please change to the IO port connected to the
RES_N pin of the RRH46410 sensor module on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 /* Delay 1s after power-on and reset */
 R_BSP_SoftwareDelay(RRH46410_WAIT_1000_MS, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_RRH46410_Open(&g_rrh46410_ctrl, &g_rrh46410_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if RRH46410_IRQ_ENABLE
    g_rrh46410_irq_callback_flag = 0;
#endif
```

```
    g_rrh46410_i2c_callback_flag = 0;
    err = RM_RRH46410_MeasurementStart(&g_rrh46410_ctrl);
    handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
    {
    }
    g_rrh46410_i2c_callback_flag = 0;
 while (1)
    {
 /* Set the current humidity */
       err = RM_RRH46410_TemperatureAndHumiditySet(&g_rrh46410_ctrl, 0.0F, humidity);
     handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
#if RRH46410_IRQ_ENABLE
 while (0U == g_rrh46410_irq_callback_flag)
     {
     }
     g_rrh46410_irq_callback_flag = 0;
#else
 /* Delay required time. See Table 3 in the RRH46410 Programming Manual. */
 R_BSP_SoftwareDelay(RRH46410_WAIT_3000_MS, BSP_DELAY_UNITS_MILLISECONDS);
#endif
 do
     {
           err = RM_RRH46410_Read(&g_rrh46410_ctrl, &raw_data);
     handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
           err = RM_RRH46410_Iaq2ndGenDataCalculate(&g_rrh46410_ctrl, &raw_data,
```

```
&rrh46410_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 /* Measurement is not complete */
 R_BSP_SoftwareDelay(RRH46410_WAIT_10_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to rrh46410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

IAQ 2nd Gen. ULP

```
void rm_rrh46410_iaq_2nd_gen_ulp_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t rrh46410_data;
 float humidity = RRH46410_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_rrh46410_cfg.p_comms_instance->p_cfg->p_extend;
```

```
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
```

```
 /* Reset sensor module (active low). Please change to the IO port connected to the
RES_N pin of the RRH46410 sensor module on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 /* Delay 1s after power-on and reset */
 R_BSP_SoftwareDelay(RRH46410_WAIT_1000_MS, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_RRH46410_Open(&g_rrh46410_ctrl, &g_rrh46410_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if RRH46410_IRQ_ENABLE
    g_rrh46410_irq_callback_flag = 0;
#endif
    g_rrh46410_i2c_callback_flag = 0;
    err = RM_RRH46410_MeasurementStart(&g_rrh46410_ctrl);
    handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
    {
    }
    g_rrh46410_i2c_callback_flag = 0;
 while (1)
    {
 /* Set the current humidity */
       err = RM_RRH46410_TemperatureAndHumiditySet(&g_rrh46410_ctrl, 0.0F, humidity);
      handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
#if RRH46410_IRQ_ENABLE
 while (0U == g_rrh46410_irq_callback_flag)
```

```
     {
     }
     g_rrh46410_irq_callback_flag = 0;
#else
 /* Delay required time. See Table 3 in the RRH46410 Programming Manual. */
 R_BSP_SoftwareDelay(RRH46410_WAIT_90000_MS, BSP_DELAY_UNITS_MILLISECONDS);
#endif
 do
     {
           err = RM_RRH46410_Read(&g_rrh46410_ctrl, &raw_data);
     handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
           err = RM_RRH46410_Iaq2ndGenDataCalculate(&g_rrh46410_ctrl, &raw_data,
&rrh46410_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 /* Measurement is not complete */
 R_BSP_SoftwareDelay(RRH46410_WAIT_10_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
 if (FSP_SUCCESS == err)
     {
 /* Describe the process by referring to rrh46410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
```

```
      }
    }
}
```

PBAQ

```
void rm_rrh46410_pbaq_basic_example (void)
{
 fsp_err_t               err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t raw_data;
 rm_zmod4xxx_pbaq_data_t rrh46410_data;
 float humidity = RRH46410_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_rrh46410_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
```

```
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset sensor module (active low). Please change to the IO port connected to the
RES_N pin of the RRH46410 sensor module on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 /* Delay 1s after power-on and reset */
 R_BSP_SoftwareDelay(RRH46410_WAIT_1000_MS, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_RRH46410_Open(&g_rrh46410_ctrl, &g_rrh46410_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if RRH46410_IRQ_ENABLE
    g_rrh46410_irq_callback_flag = 0;
#endif
    g_rrh46410_i2c_callback_flag = 0;
    err = RM_RRH46410_MeasurementStart(&g_rrh46410_ctrl);
```

```
   handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
   {
   }
   g_rrh46410_i2c_callback_flag = 0;
 while (1)
   {
 /* Set the current humidity */
       err = RM_RRH46410_TemperatureAndHumiditySet(&g_rrh46410_ctrl, 0.0F, humidity);
     handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
#if RRH46410_IRQ_ENABLE
 while (0U == g_rrh46410_irq_callback_flag)
     {
     }
     g_rrh46410_irq_callback_flag = 0;
#else
 /* Delay required time. See Table 3 in the RRH46410 Programming Manual. */
 R_BSP_SoftwareDelay(RRH46410_WAIT_5000_MS, BSP_DELAY_UNITS_MILLISECONDS);
#endif
 do
     {
           err = RM_RRH46410_Read(&g_rrh46410_ctrl, &raw_data);
     handle_error(err);
 while (0U == g_rrh46410_i2c_callback_flag)
     {
     }
     g_rrh46410_i2c_callback_flag = 0;
           err = RM_RRH46410_PbaqDataCalculate(&g_rrh46410_ctrl, &raw_data,
&rrh46410_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
```

```
     {
/* Measurement is not complete */
R_BSP_SoftwareDelay(RRH46410_WAIT_10_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
if (FSP_SUCCESS == err)
     {
/* Describe the process by referring to rrh46410_data */
     }
else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
/* Gas data is invalid. */
     }
else
     {
     handle_error(err);
     }
   }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_rrh46410_init_process_params_t |
| struct | rm_rrh46410_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_rrh46410_init_process_params_t

| struct rm_rrh46410_init_process_params_t | | |
|---|---|---|
| RRH46410 initialization process block | | |
| Data Fields | | |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile rm_zmod4xxx_event_t | event | Callback event. |

### ◆ rm_rrh46410_instance_ctrl_t

| struct rm_rrh46410_instance_ctrl_t |
|---|

RRH46410 control block

| **Data Fields** | |
|---|---|
| uint32_t | open |
| | Open flag. |
| | |
| uint8_t | write_buf [RM_RRH46410_MAX_I2C_BUF_SIZE] |
| | Write buffer for I2C communications. |
| | |
| uint8_t | read_buf [RM_RRH46410_MAX_I2C_BUF_SIZE] |
| | Read buffer for I2C communications. |
| | |
| uint8_t * | p_read_data |
| | Pointer to read data. This is used for checking error code and checksum in callback. |
| | |
| uint8_t | read_bytes |
| | Read bytes. This is used for checking error code and checksum in callback. |
| | |
| volatile uint8_t | prev_sample_id |
| | Previous sample ID. This is used for checking if sensor is in stabilization. |
| | |
| volatile int16_t | warmup_counts |
| | Counts for warning up. This is used for checking if sensor is in stabilization. |
| | |
| volatile rm_zmod4xxx_event_t | event |
| | Callback event. |
| | |

| | |
|---|---|
| rm_rrh46410_init_process_params_t | init_process_params |
| | For the initialization process. |
| | |
| rm_zmod4xxx_cfg_t const * | p_cfg |
| | Pointer of configuration block. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void const * | p_irq_instance |
| | Pointer to IRQ instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_comms_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |
| void(* | p_irq_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | IRQ callback. |
| | |

## Function Documentation

### ◆ RM_RRH46410_Open()

fsp_err_t RM_RRH46410_Open ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_cfg_t const *const *p_cfg* )

Opens and configures the RRH46410 sensor module. Implements rm_zmod4xxx_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RRH46410 successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH46410_Close()

fsp_err_t RM_RRH46410_Close ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when close the sensor module. Implements rm_zmod4xxx_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_RRH46410_MeasurementStart()

fsp_err_t RM_RRH46410_MeasurementStart ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH46410_MeasurementStop()

fsp_err_t RM_RRH46410_MeasurementStop ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully stopped. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH46410_Read()

fsp_err_t RM_RRH46410_Read ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data* )

This function should be called to get measurement results after measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH46410_Iaq2ndGenDataCalculate()

fsp_err_t RM_RRH46410_Iaq2ndGenDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_iaq_2nd_data_t *const *p_rrh46410_data* )

This function should be called when calculating gas data from measurement results. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully gas data is calculated. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_UNSUPPORTED | Unsupported operation mode. |
| FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED | Measurement is not finished. |

### ◆ RM_RRH46410_PbaqDataCalculate()

fsp_err_t RM_RRH46410_PbaqDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_pbaq_data_t *const *p_rrh46410_data* )

This function should be called when calculating gas data from measurement results. Implements rm_zmod4xxx_api_t::pbaqDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully gas data is calculated. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_UNSUPPORTED | Unsupported operation mode. |
| FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED | Measurement is not finished. |

### ◆ RM_RRH46410_TemperatureAndHumiditySet()

| fsp_err_t RM_RRH46410_TemperatureAndHumiditySet ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, float *temperature*, float *humidity* ) |
|---|
| This function should be called before Read. Humidity measurements are needed for ambient compensation. temperature is not supported. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully humidity is set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

## 5.2.16.5 RRH47000 NDIR CO2 Sensor Module (rm_rrh47000) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_RRH47000_Open (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_cfg_t const *const p_cfg)<br>Opens and configures the RRH47000 sensor module. Implements rm_air_sensor_api_t::open. More... |
| fsp_err_t | RM_RRH47000_Close (rm_air_sensor_ctrl_t *const p_api_ctrl)<br>This function should be called when close the sensor module. Implements rm_air_sensor_api_t::close. More... |
| fsp_err_t | RM_RRH47000_Read (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_raw_data_t *const p_raw_data)<br>This function should be called to get measurement results after measurement finishes. Implements rm_air_sensor_api_t::read. More... |
| fsp_err_t | RM_RRH47000_DataCalculate (rm_air_sensor_ctrl_t *const p_api_ctrl, const rm_air_sensor_raw_data_t *const p_raw_data, rm_air_sensor_data_t *const p_rrh47000_data)<br>This function should be called when calculating gas data from measurement results. Implements rm_air_sensor_api_t::dataCalculate. More... |

| | |
|---|---|
| fsp_err_t | RM_RRH47000_FirmwareVersionGet (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_version_t *const p_version)<br>This function should be called to get firmware version. Firmware version will be set "ascii_code" of p_version. Implements rm_air_sensor_api_t::firmwareVersionGet. More... |
| fsp_err_t | RM_RRH47000_StatusCheck (rm_air_sensor_ctrl_t *const p_api_ctrl)<br>StatusCheck API is unsupported. Implements rm_air_sensor_api_t::statusCheck. More... |
| fsp_err_t | RM_RRH47000_AlgorithmVersionGet (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_version_t *const p_version)<br>AlgorithmVersionGet API is unsupported. Implements rm_air_sensor_api_t::statusCheck. More... |

### Detailed Description

Middleware to implement the RRH47000 sensor module interface. This module implements the SENSOR Middleware Interface.

# Overview

RRH47000 is EOL.

This module provides an API for configuring and controlling the RRH47000 sensor module.

I2C communication with the RRH47000 sensor module is realized by connecting with the rm_comms_i2c module.

### Features

The RRH47000 sensor interface implementation has the following key features:

- Initialize the sensor for measurement
- Get the measruement results from the sensor module

# Configuration

### Build Time Configurations for rm_rrh47000

The following build time configurations are defined in fsp_cfg/rm_rrh47000_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled | Default (BSP) | If selected code for parameter checking is |

| | | |
|---|---|---|
| • Disabled | | included in the build. |

### Configurations for Sensor > RRH47000 NDIR CO2 Sensor Module (rm_rrh47000) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > RRH47000 NDIR CO2 Sensor Module (rm_rrh47000) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rrh47000_sensor0 | Module name. |
| Automatic Baseline Correction (ABC) | • Enable all<br>• Enable only ABC<br>• Enable only fast ABC<br>• Disable all | Enable all | Set Automatic Baseline Correction (ABC). |
| Correction cycle period [days] | Value must be 1 or more | 7 | Set correction cycle period. Min = 1 and Max = 30. |
| CO2 baseline value [ppm] | Value must be 400 or more | 400 | Set CO2 baseline value. Min = 400 and Max = 1500. |
| Comms I2C Callback | Name must be a valid C symbol | rrh47000_comms_i2c_callback | A user COMMS I2C callback function can be provided. |

#### Pin Configuration

This module use SDA and SCL pins of I2C Master, SCI I2C and IICA Master.

## Usage Notes

RRH47000 datasheet is here.
The RRH47000 sensor module uses nondispersive infrared (NDIR) technology to accurately measure CO2 in a compact package size while ensuring competitive pricing. An integrated relative humidity and temperature sensor enables calibration and compensation of the NDIR sensor for precise operation in different environments. Sensors for carbon dioxide (CO2, not estimated), humidity (RH), and temperature (T), are combined in a single package.
If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

#### Notifications

- This module supports I2C only (not support UART).
- The RRH47000 sensor module needs a minimum of 3 seconds to initiate communication via I²C after power-on.
- The RRH47000 sensor module supports a clock frequency ranging from 10kHz to 100kHz.

### Limitations

- I2C Master (r_sau_i2c) is not supported. RRH47000 needs clock stretching, but is not supported by SAU I2C.
- The following commands are currently not supported.
  - Start CO2 Correction
  - Get Serial Number

### Bus Initialization

The RRH47000 interface expects a bus instance to be opened before opening any RRH47000 device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

### Initialization

Initialize with RM_RRH47000_Open().

### From measurement start to data acquisition

An endless loop continuously checks the status of the RRH47000 sensor module and reads its data. The measurement results are subsequently processed, and the air quality values are calculated.

1. Call RM_RRH47000_Read() and read the measurement data.
2. Wait until RM_AIR_SENSOR_EVENT_SUCCESS is received.
3. Call RM_RRH47000_DataCalculate().

## Examples

### Basic Example

These are basic examples of minimal use of RRH47000 sensor implementation in an application.

```
void rm_rrh47000_basic_example (void)
{
 fsp_err_t                err = FSP_SUCCESS;
 rm_air_sensor_raw_data_t raw_data;
 rm_air_sensor_data_t     rrh47000_data;
 bool calculated = false;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_rrh47000_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
```

```
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Delay 3 seconds after power-on */
 R_BSP_SoftwareDelay(RRH47000_WAIT_3000_MS, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_RRH47000_Open(&g_rrh47000_ctrl, &g_rrh47000_cfg);
```

```
/* Handle any errors. This function should be defined by the user. */
   handle_error(err);
while (1)
   {
do
     {
           g_rrh47000_i2c_callback_flag = 0;
/* Read measurement results from RRH47000 */
RM_RRH47000_Read(&g_rrh47000_ctrl, &raw_data);
while (0 == g_rrh47000_i2c_callback_flag)
     {
/* Wait callback */
     }
/* Calculate CO2, Humidity and Temperatuere values from measurement results */
           err = RM_RRH47000_DataCalculate(&g_rrh47000_ctrl, &raw_data,
&rrh47000_data);
if (FSP_SUCCESS == err)
     {
/* valid data */
               calculated = 1;
     }
else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
/* invalid data. checksum error occurs. */
               calculated = 0;
     }
else
     {
     handle_error(err);
     }
     } while (0 == calculated);
/* Wait 1 seconds. */
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
   }
```

```
}
```

### Data Structures

| | |
|---|---|
| struct | rm_rrh47000_init_process_params_t |
| struct | rm_rrh47000_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_rrh47000_abc_t |

## Data Structure Documentation

### ◆ rm_rrh47000_init_process_params_t

| struct rm_rrh47000_init_process_params_t | | |
|---|---|---|
| RRH47000 initialization process block | | |
| Data Fields | | |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile rm_air_sensor_event_t | event | Callback event. |

### ◆ rm_rrh47000_instance_ctrl_t

| struct rm_rrh47000_instance_ctrl_t | |
|---|---|
| RRH47000 control block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| uint8_t | write_buf [7] |
| | Write buffer for I2C communications. |
| | |
| uint8_t | read_buf [13] |
| | Read buffer for I2C communications. |
| | |
| uint8_t * | p_read_data |
| | Pointer to read data. This is used for checksum in callback. |

| | |
|---|---|
| | |
| uint8_t | read_bytes |
| | Read bytes. This is used for checksum in callback. |
| | |
| rm_air_sensor_version_t * | p_version |
| | Pointer to firmware version. |
| | |
| volatile rm_air_sensor_event_t | event |
| | Callback event. |
| | |
| rm_rrh47000_init_process_params_t | init_process_params |
| | For the initialization process. |
| | |
| rm_air_sensor_cfg_t const * | p_cfg |
| | Pointer of configuration block. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_comms_callback )(rm_air_sensor_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |

## Enumeration Type Documentation

### ◆ rm_rrh47000_abc_t

| enum rm_rrh47000_abc_t | |
|---|---|
| RRH47000 Automatic Baseline Correction (ABC) setting. | |
| Enumerator | |
| RM_RRH47000_ABC_ENABLE_ALL | Enable all. |
| RM_RRH47000_ABC_ENABLE_ONLY_ABC | Enable only ABC (automatic baseline correction) |
| RM_RRH47000_ABC_ENABLE_ONLY_FAST_ABC | Enable only fast ABC (fast automatic baseline correction) |
| RM_RRH47000_ABC_DISABLE_ALL | Disable all. |

## Function Documentation

### ◆ RM_RRH47000_Open()

fsp_err_t RM_RRH47000_Open ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_cfg_t const *const *p_cfg* )

Opens and configures the RRH47000 sensor module. Implements rm_air_sensor_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RRH47000 successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_INVALID_DATA | Checksum error occurs. |

### ◆ RM_RRH47000_Close()

fsp_err_t RM_RRH47000_Close ( rm_air_sensor_ctrl_t *const *p_api_ctrl*)

This function should be called when close the sensor module. Implements rm_air_sensor_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_RRH47000_Read()

fsp_err_t RM_RRH47000_Read ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_raw_data_t *const *p_raw_data* )

This function should be called to get measurement results after measurement finishes. Implements rm_air_sensor_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH47000_DataCalculate()

fsp_err_t RM_RRH47000_DataCalculate ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, const rm_air_sensor_raw_data_t *const *p_raw_data*, rm_air_sensor_data_t *const *p_rrh47000_data* )

This function should be called when calculating gas data from measurement results. Implements rm_air_sensor_api_t::dataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully gas data is calculated. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_SENSOR_INVALID_DATA | Checksum error occurs. |

### ◆ RM_RRH47000_FirmwareVersionGet()

fsp_err_t RM_RRH47000_FirmwareVersionGet ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_version_t *const *p_version* )

This function should be called to get firmware version. Firmware version will be set "ascii_code" of p_version. Implements rm_air_sensor_api_t::firmwareVersionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_INVALID_POINTER | Pointer is NULL. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH47000_StatusCheck()

fsp_err_t RM_RRH47000_StatusCheck ( rm_air_sensor_ctrl_t *const *p_api_ctrl*)

StatusCheck API is unsupported. Implements rm_air_sensor_api_t::statusCheck.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

### ◆ RM_RRH47000_AlgorithmVersionGet()

fsp_err_t RM_RRH47000_AlgorithmVersionGet ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_version_t *const *p_version* )

AlgorithmVersionGet API is unsupported. Implements rm_air_sensor_api_t::statusCheck.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Unsupported. |

## 5.2.16.6 RRH62000 All-in-one Air Quality Module (rm_rrh62000) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_RRH62000_Open (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_cfg_t const *const p_cfg)<br><br>Opens and configures the RRH62000 sensor module. If ZMOD cleaning is not complete, this function will take at least 1 minute. Implements rm_air_sensor_api_t::open. More... |
| fsp_err_t | RM_RRH62000_Close (rm_air_sensor_ctrl_t *const p_api_ctrl)<br><br>This function should be called when close the sensor module. Implements rm_air_sensor_api_t::close. More... |
| fsp_err_t | RM_RRH62000_StatusCheck (rm_air_sensor_ctrl_t *const p_api_ctrl)<br><br>This function should be called when polling is used. It reads the status of sensor. Implements rm_air_sensor_api_t::statusCheck. More... |
| fsp_err_t | RM_RRH62000_Read (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_raw_data_t *const p_raw_data)<br><br>This function should be called to get measurement results after measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_air_sensor_api_t::read. More... |
| fsp_err_t | RM_RRH62000_DataCalculate (rm_air_sensor_ctrl_t *const p_api_ctrl, const rm_air_sensor_raw_data_t *const p_raw_data, rm_air_sensor_data_t *const p_rrh62000_data)<br><br>This function should be called when calculating gas data from measurement results. Implements rm_air_sensor_api_t::dataCalculate. More... |
| fsp_err_t | RM_RRH62000_FirmwareVersionGet (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_version_t *const p_version)<br><br>This function should be called to get firmware version. Firmware version does not use "patch" of p_version. Implements rm_air_sensor_api_t::firmwareVersionGet. More... |
| fsp_err_t | RM_RRH62000_AlgorithmVersionGet (rm_air_sensor_ctrl_t *const p_api_ctrl, rm_air_sensor_version_t *const p_version)<br><br>This function should be called to get algorithm version. Implements rm_air_sensor_api_t::algorithmVersionGet. More... |

### Detailed Description

Middleware to implement the RRH62000 sensor module interface. This module implements the SENSOR Middleware Interface.

# Overview

RRH62000 is EOL.

This module provides an API for configuring and controlling the RRH62000 sensor module.

I2C communication with the RRH62000 sensor module is realized by connecting with the rm_comms_i2c module.

### Features

The RRH62000 sensor interface implementation has the following key features:

- Initialize the sensor for measurement
- Receive status for wait until the measurement is done.
- Get the measruement results from the sensor module

# Configuration

### Build Time Configurations for rm_rrh62000

The following build time configurations are defined in fsp_cfg/rm_rrh62000_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Sensor > RRH62000 All-in-one Air Quality Module (rm_rrh62000) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > RRH62000 All-in-one Air Quality Module (rm_rrh62000) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rrh62000_sensor0 | Module name. |
| Number of moving average [times] | Value must be 1 or more | 10 | Set number of moving average. Min = 1 and Max = 60. |
| Fan speed control [%] | Value must be 60 or more | 86 | Set fan speed control. Min = 60 and Max = 100. |
| Comms I2C Callback | Name must be a valid C symbol | rrh62000_comms_i2c_callback | A user COMMS I2C callback function can |

be provided.

### Pin Configuration

This module use SDA and SCL pins of I2C Master, SCI I2C and IICA Master.

# Usage Notes

RRH62000 datasheet is here.
When using the sensor for the first time, allow up to 48 hours for re-hydration of the humidity sensor in ambient air and for TVOC sensor conditioning. This will allow the module to take optimal readings.
The RRH62000 is an integrated sensor module for the measurement of critical air quality parameters. Sensors for particulate matter (PM), total volatile organic compounds (TVOC), estimation of carbon dioxide (eCO2) optional nitrogen dioxide (NO2), optional ozone (O3), humidity (RH), and temperature (T), are combined in a single package.
If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

### Notifications

- After turning on the module for the first time, the sensor will automatically start in ZMOD sensor cleaning mode for 1 minute. Do not cut power to the sensor during this period.
- Interval time needs to be at least 1s after power-on or reset.
- I2C-master access interval for RRH62000 must be longer than 1s.
- This module supports I2C only (not support UART).

### Limitations

- r_iic_master on RA8 series MCU is not supported.
- I2C Master (r_sau_i2c) is not supported. RRH62000 needs clock stretching, but is not supported by SAU I2C.
- The following commands are currently not supported.
    - SLEEP
    - CLEAN
    - TINTC_H
    - TINTC_L
    - TCLEAN
    - RMOX
    - UID

### Bus Initialization

The RRH62000 interface expects a bus instance to be opened before opening any RRH62000 device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

### Initialization

Initialize with RM_RRH62000_Open().

### From measurement start to data acquisition

An endless loop continuously checks the status of the RRH62000 sensor module and reads its data. The measurement results are subsequently processed, and the air quality values are calculated.

1. Call RM_RRH62000_StatusCheck(). This function will execute a status check over I2C.
2. If RM_AIR_SENSOR_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback, user should wait some time (min=1s) and then call RM_RRH62000_StatusCheck() again.
3. Wait until RM_AIR_SENSOR_EVENT_MEASUREMENT_COMPLETE is received.
4. Call RM_RRH62000_Read() and read the measurement data.
5. Wait until RM_AIR_SENSOR_EVENT_SUCCESS is received.
6. Call RM_RRH62000_DataCalculate().

# Examples

### Basic Example

These are basic examples of minimal use of RRH62000 sensor implementation in an application.

```
void rm_rrh62000_basic_example (void)
{
 fsp_err_t                err = FSP_SUCCESS;
 rm_air_sensor_raw_data_t raw_data;
 rm_air_sensor_data_t     rrh62000_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_rrh62000_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
```

```
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Delay 1s after power-on */
 R_BSP_SoftwareDelay(RRH62000_WAIT_1000_MS, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_RRH62000_Open(&g_rrh62000_ctrl, &g_rrh62000_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    g_rrh62000_i2c_callback_flag = 0;
 while (1)
    {
 do
     {
            err = RM_RRH62000_StatusCheck(&g_rrh62000_ctrl);
     handle_error(err);
 while (0U == g_rrh62000_i2c_callback_flag)
     {
```

```
     }
     g_rrh62000_i2c_callback_flag = 0;
           err = RM_RRH62000_Read(&g_rrh62000_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(RRH62000_WAIT_1000_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_rrh62000_i2c_callback_flag)
     {
     }
     g_rrh62000_i2c_callback_flag = 0;
      err = RM_RRH62000_DataCalculate(&g_rrh62000_ctrl, &raw_data, &rrh62000_data);
     handle_error(err);
   }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_rrh62000_init_process_params_t |
| struct | rm_rrh62000_status_params_t |
| struct | rm_rrh62000_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_rrh62000_init_process_params_t

| struct rm_rrh62000_init_process_params_t | | |
|---|---|---|
| RRH62000 initialization process block | | |
| Data Fields | | |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile rm_air_sensor_event_t | event | Callback event. |

### ◆ rm_rrh62000_status_params_t

| struct rm_rrh62000_status_params_t |
|---|
| RRH62000 status parameters block |

| Data Fields | | |
|---|---|---|
| uint8_t | value | Value to check measurement complete. |
| volatile bool | flag | Flag to identify StatusCheck API and other APIs in callback. |

### ◆ rm_rrh62000_instance_ctrl_t

| struct rm_rrh62000_instance_ctrl_t | |
|---|---|
| RRH62000 control block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| uint8_t | write_buf [2] |
| | Write buffer for I2C communications. |
| | |
| uint8_t | read_buf [2] |
| | Read buffer for I2C communications. |
| | |
| uint8_t * | p_read_data |
| | Pointer to read data. This is used for checking error code and checksum in callback. |
| | |
| uint8_t | read_bytes |
| | Read bytes. This is used for checking error code and checksum in callback. |
| | |
| volatile rm_air_sensor_event_t | event |
| | Callback event. |
| | |
| rm_rrh62000_status_params_t | status |

| | |
|---|---|
| | Status parameter for arriving new measurement results. |
| | |
| rm_rrh62000_status_params_t | cleaning_status |
| | Status parameter for ZMOD cleaning. |
| | |
| rm_rrh62000_init_process_params_t | init_process_params |
| | For the initialization process. |
| | |
| rm_air_sensor_cfg_t const * | p_cfg |
| | Pointer of configuration block. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_comms_callback )(rm_air_sensor_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |

## Function Documentation

### ◆ RM_RRH62000_Open()

fsp_err_t RM_RRH62000_Open ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_cfg_t const *const *p_cfg* )

Opens and configures the RRH62000 sensor module. If ZMOD cleaning is not complete, this function will take at least 1 minute. Implements rm_air_sensor_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | RRH62000 successfully configured. |
| FSP_ERR_ASSERTION | Null pointer, or one or more configuration options is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. This module can only be opened once. |
| FSP_ERR_NOT_INITIALIZED | ZMOD cleaning is not complete. |

### ◆ RM_RRH62000_Close()

fsp_err_t RM_RRH62000_Close ( rm_air_sensor_ctrl_t *const *p_api_ctrl*)

This function should be called when close the sensor module. Implements rm_air_sensor_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_RRH62000_StatusCheck()

fsp_err_t RM_RRH62000_StatusCheck ( rm_air_sensor_ctrl_t *const *p_api_ctrl*)

This function should be called when polling is used. It reads the status of sensor. Implements rm_air_sensor_api_t::statusCheck.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_RRH62000_Read()

fsp_err_t RM_RRH62000_Read ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_raw_data_t *const *p_raw_data* )

This function should be called to get measurement results after measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_air_sensor_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH62000_DataCalculate()

fsp_err_t RM_RRH62000_DataCalculate ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, const rm_air_sensor_raw_data_t *const *p_raw_data*, rm_air_sensor_data_t *const *p_rrh62000_data* )

This function should be called when calculating gas data from measurement results. Implements rm_air_sensor_api_t::dataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully gas data is calculated. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_RRH62000_FirmwareVersionGet()

fsp_err_t RM_RRH62000_FirmwareVersionGet ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_version_t *const *p_version* )

This function should be called to get firmware version. Firmware version does not use "patch" of p_version. Implements rm_air_sensor_api_t::firmwareVersionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

### ◆ RM_RRH62000_AlgorithmVersionGet()

fsp_err_t RM_RRH62000_AlgorithmVersionGet ( rm_air_sensor_ctrl_t *const *p_api_ctrl*, rm_air_sensor_version_t *const *p_version* )

This function should be called to get algorithm version. Implements rm_air_sensor_api_t::algorithmVersionGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |

## 5.2.16.7 ZMOD4XXX Gas Sensor (rm_zmod4xxx) [Deprecated]

Modules » Sensor

### Functions

| | |
|---|---|
| fsp_err_t | RM_ZMOD4XXX_Open (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg)<br><br>This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the zmod4xxx and start a measurement. Implements |

| | |
|---|---|
| | rm_zmod4xxx_api_t::open. More... |
| fsp_err_t | RM_ZMOD4XXX_Close (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when close the sensor. Implements rm_zmod4xxx_api_t::close. More... |
| fsp_err_t | RM_ZMOD4XXX_MeasurementStart (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart. More... |
| fsp_err_t | RM_ZMOD4XXX_MeasurementStop (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop. More... |
| fsp_err_t | RM_ZMOD4XXX_StatusCheck (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br>This function should be called when polling is used. It reads the status of sensor. Implements rm_zmod4xxx_api_t::statusCheck. More... |
| fsp_err_t | RM_ZMOD4XXX_Read (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data)<br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read. More... |
| fsp_err_t | RM_ZMOD4XXX_Iaq1stGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data)<br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq1stGenDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_Iaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data)<br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate. More... |

| | |
|---|---|
| fsp_err_t | RM_ZMOD4XXX_SulfurOdorDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::sulfurOdorDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_Oaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq2ndGenDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_RaqDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_raq_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::raqDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_RelIaqDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_rel_iaq_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::relIaqDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_PbaqDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_pbaq_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::pbaqDataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_No2O3DataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_no2_o3_data_t *const p_zmod4xxx_data)<br><br>This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::no2O3DataCalculate. More... |
| fsp_err_t | RM_ZMOD4XXX_CompensationInfoSet (rm_zmod4xxx_ctrl_t *const |

| | |
|---|---|
| | p_api_ctrl, rm_zmod4xxx_compensation_info_t const *const p_compensation_info)<br><br>This function should be called when setting the compensation info from ZMOD4510 device for the ZMOD4410 device and when the ZMOD4510 for oxidizing gas correction option is enabled. After reading the raw data from the ZMOD4510 device, set the compensation info for the ZMOD4410 to begin calculating the compensation data. More... |
| fsp_err_t | RM_ZMOD4XXX_TemperatureAndHumiditySet (rm_zmod4xxx_ctrl_t *const p_api_ctrl, float temperature, float humidity)<br><br>This function is valid only for OAQ_2nd_Gen and IAQ_2nd_Gen_ULP. This function should be called before DataCalculate. Humidity and temperature measurements are needed for ambient compensation. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet. More... |
| fsp_err_t | RM_ZMOD4XXX_DeviceErrorCheck (rm_zmod4xxx_ctrl_t *const p_api_ctrl)<br><br>This function is valid only for IAQ_2nd_Gen and IAQ_2nd_Gen_ULP. This function should be called before Read and DataCalculate. Check for unexpected reset occurs or getting unvalid ADC data. Implements rm_zmod4xxx_api_t::deviceErrorCheck. More... |

### Detailed Description

Middleware to implement the ZMOD4XXX sensor interface. This module implements the ZMOD4XXX Middleware Interface.

# Overview

ZMOD4410, ZMOD4450 and ZMOD4510 are EOL.

This module provides an API for configuring and controlling the ZMOD4XXX sensor. Supported ZMOD4XXX sensors are below.

- ZMOD4410
- ZMOD4450
- ZMOD4510

I2C communication with the ZMOD4XXX sensor is realized by connecting with the rm_comms_i2c module.

### Features

The ZMOD4XXX sensor interface implementation has the following key features:

- Initialize the sensor for measurement

- Start a measurement at any time
- Read status register for wait until the measurement is done. This will also be signaled by an interrupt
- Get the ADC data from the sensor
- Input the ADC data and acquire the air quality values by calculation in the library.

# Configuration

### Build Time Configurations for rm_zmod4xxx

The following build time configurations are defined in fsp_cfg/rm_zmod4xxx_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Sensor > ZMOD4XXX Gas Sensor (rm_zmod4xxx) [Deprecated]

This module can be added to the Stacks tab via New Stack > Sensor > ZMOD4XXX Gas Sensor (rm_zmod4xxx) [Deprecated].

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_zmod4xxx_sensor0 | Module name. |
| Comms I2C Callback | Name must be a valid C symbol | zmod4xxx_comms_i2c_callback | A user COMMS I2C callback function can be provided. |
| IRQ Callback | Name must be a valid C symbol | zmod4xxx_irq_callback | A user IRQ callback function can be provided. |

### Pin Configuration

This module use SDA and SCL pins of I2C Master and SCI I2C.

# Usage Notes

ZMOD4410 datasheet is here.
The ZMOD4410 has five modes of operation.
The ZMOD4410 will respond to TVOC immediately upon start-up; however, a conditioning period of 48 hours followed by a sensor module restart in an ambient environment is recommended to improve stability and obtain maximum performance.
Best results are achieved with continuous operation because the module algorithm can learn about the environment over time.

| Method | Description |
|---|---|
| IAQ 1st Generation Continuous | Measurement of UBA levels for IAQ and eCO2, provides continuous data |

| | |
|---|---|
| IAQ 1st Generation Low Power | Measurement of UBA levels for IAQ and eCO2, fixed sampling interval of 6 seconds |
| IAQ 2nd Generation | Using AI for improved ppm TVOC, IAQ and eCO2 functionality (recommended for new designs) |
| IAQ 2nd Generation Ultra Low Power | Using AI for improved ppm TVOC, IAQ and eCO2 functionality. This method offers a much lower power consumption |
| Relative IAQ | Lightweight algorithm reacts to air quality changes and outputs a relative IAQ |
| Relative IAQ Ultra Low Power | Lightweight algorithm reacts to air quality changes and outputs a relative IAQ. This method offers a much lower power consumption |
| Public Building AQ Standard (PBAQ) | For highly accurate and consistent sensor readings to fulfill public building standards. |
| Sulfur-based Odor Discrimination | The odors in "sulfur" (sulfur based) and "acceptable" (organic based) and shows an intensity level of the smell |

ZMOD4450 datasheet is here.
The ZMOD4450 has one modes of operation.
The response time for a gas stimulation is always within a few seconds, depending on the gas and its concentration. An active or direct airflow onto the sensor module is not necessary since diffusion of ambient gas does not limit the sensor module response time. The ZMOD4450 will respond to typical refrigeration gases immediate upon start-up; however, a conditioning period of 48 hours in a refrigeration environment is recommended to improve stability and get maximum performance, as the module algorithm is able to learn about the refrigeration environment over time.

| Method | Description |
|---|---|
| RAQ | Control signal based on Refrigerator Air Quality Changes |

ZMOD4510 datasheet is here.
The ZMOD4510 has two modes of operation.
For OAQ 2nd Gen operation a response to ozone will be seen after a warmup time of 30 min.
For NO2 O3 operation a response to nitrogen_dioxide and ozone will be seen after a warmup time of 5 min.
For Compensation for oxidizing gases ULP option operation a response to ozone will be seen after a warmup time of 30 min.
For Compensation for oxidizing gases option operation a response to nitrogen_dioxide and ozone will be seen after a warmup time of 5 min.
In all operation modes a conditioning period of 48 hours followed by a sensor module restart in an ambient environment is recommended to improve stability and obtain maximum performance.

| Method | Description |
|---|---|
| OAQ 2nd Generation | Selective Ozone featuring Ultra-Low Power |
| NO2 O3 | Selective nitrogen dioxide and ozone firmware |
| Compensation for oxidizing gases option | Measurement of compensation data for IAQ 2nd |

| | |
|---|---|
| | Gen, PBAQ |
| Compensation for oxidizing gases option ULP | Measurement of compensation data for IAQ 2nd Gen Ultra-Low Power |

A library corresponding to each of these modes is required.
The **Compensation for oxidizing gases option** mode can be used with the **IAQ 2nd Generation** mode and the **Public Building AQ Standard (PBAQ)** mode.
The **Compensation for oxidizing gases option ULP** mode can be used with the **IAQ 2nd Generation Ultra Low Power** mode.
By setting in RA Configuration, the library will be generated in the ra/fsp/lib/rm_zmod4xxx folder of your project.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

**Notifications**

The ZMOD4xxx sensor needs a reset before operation; please input a reset signal to the RES_N pin (active low).

Program flow for OAQ 2nd gen is changed in FSP v4.0.0. Please refer to the programming manual [R36US0004EU] and the application note[R01AN5899].
R01AN5899 : https://www.renesas.com/document/apn/zmod4xxx-sample-application
R36US0004EU : https://www.renesas.com/document/mat/zmod4510-programming-manual-read-me

**Limitations**

- The following modes are currently not supported for dev_assist.
  - Compensation
  - Compensation ULP

**Bus Initialization**

The ZMOD4XXX interface expects a bus instance to be opened before opening any ZMOD4XXX device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

**Initialization**

Initialize with RM_ZMOD4XXX_Open(). One channel of timer is required to measure the waiting time at initialization.

**From measurement start to data acquisition**

After normal completion, start the measurement with RM_ZMOD4XXX_MeasurementStart(). An endless loop continuously checks the status of the ZMOD4XXX sensor and reads its data. The raw data is subsequently processed, and the air quality values are calculated.

**If IRQ is enabled**

1. Call RM_ZMOD4XXX_MeasurementStart().
2. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received via IRQ callback.
3. Call RM_ZMOD4XXX_Read(). This function will read the ADC data.
4. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
5. Call the DataCalculate API according to the mode.

**If IRQ is disabled**

1. Call RM_ZMOD4XXX_MeasurementStart().
2. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
3. Call RM_ZMOD4XXX_StatusCheck(). This function will execute a status check over I2C.
4. If RM_ZMOD4XXX_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback, user should wait some time and then call RM_ZMOD4XXX_StatusCheck() again.
5. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received.
6. Call RM_ZMOD4XXX_Read() and read the ADC data.
7. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
8. Call the DataCalculate API according to the mode.

**Checking device error for IAQ 2nd Gen.**

1. Call RM_ZMOD4XXX_DeviceErrorCheck(). This function will execute a device error check over I2C.
2. If any device error occurs, RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET or RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT are received in callback. If no device error, RM_ZMOD4XXX_EVENT_SUCCESS is received in callback.

# Examples

### Basic Example

These are basic examples of minimal use of ZMOD4XXX sensor implementation in an application.

IAQ 1st Gen. Continuous mode

```
void rm_zmod4xxx_iaq_1st_gen_continuous_basic_example (void)
{
 fsp_err_t                 err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_1st_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
```

```
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
```

```
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if ZMOD4XXX_IRQ_ENABLE
    g_zmod4xxx_irq_callback_flag = 0;
#endif
    g_zmod4xxx_i2c_callback_flag = 0;
    err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
    handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
    {
    }
    g_zmod4xxx_i2c_callback_flag = 0;
 while (1)
    {
 do
     {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
     {
     }
     g_zmod4xxx_irq_callback_flag = 0;
#else
           err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
#endif
```

```
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Iaq1stGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

IAQ 1st Gen. Low Power mode

```
void rm_zmod4xxx_iaq_1st_gen_low_power_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
```

```
 rm_zmod4xxx_iaq_1st_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
```

```
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
```

```
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Iaq1stGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
```

```
 /* Delay required time. See Table 3 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_5475_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

IAQ 2nd Gen.

```
void rm_zmod4xxx_iaq_2nd_gen_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t     raw_data;
 rm_zmod4xxx_iaq_2nd_data_t zmod4410_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,
```

```
p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
```

```
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 /* Delay required time. See Table 3 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_3000_MS, BSP_DELAY_UNITS_MILLISECONDS);
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
                (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
      {
```

```
/* Error during read of sensor status. Please reset device. */
while (1)
     {
                  ;
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
/* Check validness of ADC results:
 * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
 * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
 *//* Please reset device. */
while (1)
```

```
     {
               ;
     }
     }
 /* Set the current temperature and humidity */
      err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
     handle_error(err);
      err = RM_ZMOD4XXX_Iaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
     {
 /* Describe the process by referring to zmod4410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
     }
   }
}
```

IAQ 2nd Gen Compensation.

```
void rm_zmod4xxx_iaq_2nd_gen_compensation_basic_example (void)
{
 fsp_err_t                 err = FSP_SUCCESS;
```

```
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t zmod4410_data;
    uint8_t loop_cnt = 0;
 /* ZMOD4510 Compensation info declarations */
 rm_zmod4xxx_raw_data_t          compensation_sensor_raw_data;
 rm_zmod4xxx_compensation_info_t g_zmod4xxx_compensation_info;
    g_zmod4xxx_compensation_info.p_compensation_sensor_cfg =
&compensation_sensor_cfg;
    g_zmod4xxx_compensation_info.p_raw_data               =
&compensation_sensor_raw_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
```

```
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = RM_ZMOD4XXX_Open(&compensation_sensor_ctrl, &compensation_sensor_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
 /* Trigger compensation measurement cycles every 6 seconds */
 if (loop_cnt == 0)
```

```
     {
           compensation_sensor_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_MeasurementStart(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
     }
     g_zmod4xxx_i2c_callback_flag = 0;
     err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
/* Delay required time. See Table 3 in the ZMOD4410 Programming Manual. */
R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_3000_MS, BSP_DELAY_UNITS_MILLISECONDS);
if (loop_cnt == 1)
     {
do
     {
               err = RM_ZMOD4XXX_StatusCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
             compensation_sensor_i2c_callback_flag = 0;
               err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
             compensation_sensor_i2c_callback_flag = 0;
```

```
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
                   (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
compensation_sensor_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
                    ;
     }
     }
               err = RM_ZMOD4XXX_Read(&compensation_sensor_ctrl,
&compensation_sensor_raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
compensation_sensor_i2c_callback_event))
     {
```

```
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
      {
                  ;
      }
      }
      }
 do
      {
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
      {
 /* Error during read of sensor status. Please reset device. */
 while (1)
      {
                  ;
```

```
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
     {
               ;
     }
     }
 /* Set the current temperature and humidity */
      err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
```

```
      handle_error(err);
 /* Set the compensation info */
       err = RM_ZMOD4XXX_CompensationInfoSet(&g_zmod4xxx_ctrl,
&g_zmod4xxx_compensation_info);
      handle_error(err);
       err = RM_ZMOD4XXX_Iaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
       loop_cnt += 1;
       loop_cnt %= IAQ_2ND_GEN_MAX_LOOP_CNT; // one complete measurement cycle is 2
loop counts
    }
}
```

IAQ 2nd Gen. ULP

```
void rm_zmod4xxx_iaq_2nd_gen_ulp_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
```

```
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t zmod4410_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
```

```
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 /* First delay. See Table 4 in the ZMOD4410 Programming Manual. It should be longer
than 1010 ms. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1010_MS, BSP_DELAY_UNITS_MILLISECONDS);
```

```
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
                (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
      {
 /* Error during read of sensor status or Measurement not completed due to unexpected
reset. Please reset device. */
 while (1)
      {
                  ;
      }
      }
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
```

```
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
     {
               ;
     }
     }
 /* Set the current temperature and humidity */
      err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
     handle_error(err);
```

```
      err = RM_ZMOD4XXX_Iaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
     {
 /* Describe the process by referring to zmod4410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
     }
 /* Second delay. See Table 4 in the ZMOD4410 Programming Manual. The sum of the
first and second delay should amount 90 seconds. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_90000_MS - ZMOD4XXX_WAIT_1010_MS,
BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

IAQ 2nd Gen Compensation. ULP

```
void rm_zmod4xxx_iaq_2nd_gen_ulp_compensation_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t zmod4410_data;
    uint8_t loop_cnt = 0;
    uint32_t remainingLoopTime;
```

```
 /* ZMOD4510 Compensation info declarations */
 rm_zmod4xxx_raw_data_t          compensation_sensor_raw_data;
 rm_zmod4xxx_compensation_info_t g_zmod4xxx_compensation_info;
    g_zmod4xxx_compensation_info.p_compensation_sensor_cfg =
&compensation_sensor_cfg;
    g_zmod4xxx_compensation_info.p_raw_data               =
&compensation_sensor_raw_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
```

```
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = RM_ZMOD4XXX_Open(&compensation_sensor_ctrl, &compensation_sensor_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
       remainingLoopTime = ZMOD4XXX_WAIT_2000_MS; /* loop period is 2s */
       compensation_sensor_i2c_callback_flag = 0;
 /* Start ZMOD4510 measurement every 2 seconds (needed for compensation) */
       err = RM_ZMOD4XXX_MeasurementStart(&compensation_sensor_ctrl);
      handle_error(err);
```

```
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
      compensation_sensor_i2c_callback_flag = 0;
if (loop_cnt == 0)
     {
     g_zmod4xxx_i2c_callback_flag = 0;
/* Start ZMOD4410 ULP measurement every 90 seconds in loop 0 */
     err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
     }
if (loop_cnt == 1)
     {
do
     {
               err = RM_ZMOD4XXX_StatusCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
             compensation_sensor_i2c_callback_flag = 0;
               err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
             compensation_sensor_i2c_callback_flag = 0;
if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
                  (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
```

```
compensation_sensor_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
                    ;
     }
     }
 /* Read ZMOD4510 Compensation ULP measurement result in loop 1 and assign it
  * to the ZMOD4410 algorithm input data structure.
  * IMPORTANT NOTE: If Compensation ULP calculation is required, sensor
  * data MUST be read and fed into the algorithm in each loop iteration!
  */
               err                = RM_ZMOD4XXX_Read(&compensation_sensor_ctrl,
&compensation_sensor_raw_data);
               remainingLoopTime -= 1;
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
```

```
                (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
compensation_sensor_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
     {
                 ;
     }
     }
     }
 if (loop_cnt == 1)
     {
 do
     {
             err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
             err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
                 (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
```

```
     {
 /* Error during read of sensor status or Measurement not completed due to unexpected
reset. Please reset device. */
 while (1)
     {
                    ;
     }
     }
              err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
```

```
 *//* Please reset device. */
while (1)
     {
                  ;
     }
     }
/* Set the current temperature and humidity */
        err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
     handle_error(err);
/* Set the compensation info */
        err = RM_ZMOD4XXX_CompensationInfoSet(&g_zmod4xxx_ctrl,
&g_zmod4xxx_compensation_info);
     handle_error(err);
        err = RM_ZMOD4XXX_Iaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
if (FSP_SUCCESS == err)
     {
/* Describe the process by referring to zmod4410_data */
     }
else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
/* Gas data is invalid. */
     }
else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
/* Gas data is invalid. */
     }
else
     {
     handle_error(err);
     }
     }
R_BSP_SoftwareDelay(remainingLoopTime, BSP_DELAY_UNITS_MILLISECONDS);
```

```
        loop_cnt += 1;
        loop_cnt %= IAQ_2ND_GEN_ULP_MAX_LOOP_CNT;
    }
}
```

Relative IAQ

```
void rm_zmod4xxx_rel_iaq_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_rel_iaq_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
```

```
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
```

```
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 /* Delay required time. See Table 6 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_3000_MS, BSP_DELAY_UNITS_MILLISECONDS);
 do
     {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
     {
     }
     g_zmod4xxx_irq_callback_flag = 0;
#else
           err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
#endif
           err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
```

```
     {
                 ;
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
               ;
     }
     }
      err = RM_ZMOD4XXX_RelIaqDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
```

```
     {
 /* Describe the process by referring to zmod4410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
     }
   }
}
```

Relative IAQ ULP

```
void rm_zmod4xxx_rel_iaq_ulp_basic_example (void)
{
 fsp_err_t                 err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_rel_iaq_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
```

```
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
```

```
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 /* First delay. See Table 7 in the ZMOD4410 Programming Manual. It should be longer
than 1010 ms. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1500_MS, BSP_DELAY_UNITS_MILLISECONDS);
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
```

```
     g_zmod4xxx_i2c_callback_flag = 0;
#endif
           err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status or Measurement not completed due to unexpected
reset. Please reset device. */
 while (1)
     {
                 ;
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
```

```
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
     {
               ;
     }
     }
      err = RM_ZMOD4XXX_RelIaqDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
     {
 /* Describe the process by referring to zmod4410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
```

```
     }
 /* Second delay. See Table 7 in the ZMOD4410 Programming Manual. The sum of the
first and second delay should amount 90 seconds. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_90000_MS - ZMOD4XXX_WAIT_1010_MS,
BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

PBAQ

```
void rm_zmod4xxx_pbaq_basic_example (void)
{
 fsp_err_t               err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t raw_data;
 rm_zmod4xxx_pbaq_data_t zmod4410_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
```

```
      *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
           xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                          (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
      tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                      p_extend->p_bus_recursive_mutex->p_mutex_name,
                      TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
      *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
           xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
```

```
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 /* Delay required time. See Table 5 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_5000_MS, BSP_DELAY_UNITS_MILLISECONDS);
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
```

```
                (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
                 ;
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
               ;
```

```
     }
     }
 /* Set the current temperature and humidity */
      err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
     handle_error(err);
      err = RM_ZMOD4XXX_PbaqDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
     {
 /* Describe the process by referring to zmod4410_data */
     }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
     {
 /* Gas data is invalid. */
     }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
     {
 /* Gas data is invalid. */
     }
 else
     {
     handle_error(err);
     }
   }
}
```

PBAQ Compensation

```
void rm_zmod4xxx_pbaq_compensation_basic_example (void)
{
 fsp_err_t               err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t raw_data;
 rm_zmod4xxx_pbaq_data_t zmod4410_data;
```

```
    uint8_t                loop_cnt = 0;
    uint32_t                remainingLoopTime;
 /* ZMOD4510 Compensation info declarations */
 rm_zmod4xxx_raw_data_t          compensation_sensor_raw_data;
 rm_zmod4xxx_compensation_info_t g_zmod4xxx_compensation_info;
    g_zmod4xxx_compensation_info.p_compensation_sensor_cfg =
&compensation_sensor_cfg;
    g_zmod4xxx_compensation_info.p_raw_data               =
&compensation_sensor_raw_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
```

```
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
    err = RM_ZMOD4XXX_Open(&compensation_sensor_ctrl, &compensation_sensor_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
       remainingLoopTime = ZMOD4XXX_WAIT_1000_MS; /* loop period is 1s */
 if ((loop_cnt % 6) == 0)
     {
```

```
            compensation_sensor_i2c_callback_flag = 0;
            err = RM_ZMOD4XXX_MeasurementStart(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
            compensation_sensor_i2c_callback_flag = 0;
     }
if ((loop_cnt % 5) == 0)
     {
     g_zmod4xxx_i2c_callback_flag = 0;
     err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
     }
if ((loop_cnt % 6) == 5)
     {
do
     {
               err = RM_ZMOD4XXX_StatusCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
            compensation_sensor_i2c_callback_flag = 0;
              err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
            compensation_sensor_i2c_callback_flag = 0;
```

```
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
                   (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
compensation_sensor_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
 while (1)
     {
                    ;
     }
     }
               err                = RM_ZMOD4XXX_Read(&compensation_sensor_ctrl,
&compensation_sensor_raw_data);
               remainingLoopTime -= 1;
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_DeviceErrorCheck(&compensation_sensor_ctrl);
     handle_error(err);
 while (0U == compensation_sensor_i2c_callback_flag)
     {
     }
           compensation_sensor_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET ==
compensation_sensor_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
compensation_sensor_i2c_callback_event))
```

```
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
     {
                 ;
     }
     }
     }
 if ((loop_cnt % 5) == 4)
     {
 do
     {
             err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
             err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
                 (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status. Please reset device. */
```

```
while (1)
     {
                    ;
     }
     }
              err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
           err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
/* Error during read of sensor status. Please reset device. */
while (1)
     {
                   ;
     }
     }
/* Set the current temperature and humidity */
        err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
```

```
humidity);
      handle_error(err);
 /* Set the compensation info */
         err = RM_ZMOD4XXX_CompensationInfoSet(&g_zmod4xxx_ctrl,
&g_zmod4xxx_compensation_info);
      handle_error(err);
         err = RM_ZMOD4XXX_PbaqDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
      }
 R_BSP_SoftwareDelay(remainingLoopTime, BSP_DELAY_UNITS_MILLISECONDS);
       loop_cnt += 1;
       loop_cnt %= PBAQ_MAX_LOOP_CNT;
    }
}
```

### Sulfur Odor

```
void rm_zmod4xxx_sulfur_odor_basic_example (void)
```

```
{
 fsp_err_t                      err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t         raw_data;
 rm_zmod4xxx_sulfur_odor_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
```

```
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
```

```
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_SulfurOdorDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
```

```
      {
      handle_error(err);
      }
 /* Delay required time. See Table 6 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1990_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

OAQ 2nd Gen.

```
void rm_zmod4xxx_oaq_2nd_gen_basic_example (void)
{
 fsp_err_t                 err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_oaq_2nd_data_t zmod4510_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
```

```
      *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
           xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                          (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
      tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                      p_extend->p_bus_recursive_mutex->p_mutex_name,
                      TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
      *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
           xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
```

```
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
 /* Delay required time. See Table 4 in the ZMOD4510 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_2000_MS, BSP_DELAY_UNITS_MILLISECONDS);
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
```

```
                (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Error during read of sensor status or Measurement not completed due to unexpected
reset. Please reset device. */
 while (1)
     {
                 ;
     }
     }
           err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
     {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
     }
     } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
     {
     }
     g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
           (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
     {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
```

```
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
      {
                ;
      }
      }
 /* Set the current temperature and humidity */
       err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
humidity);
      handle_error(err);
       err = RM_ZMOD4XXX_Oaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4510_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4510_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

NO2 O3

```
void rm_zmod4xxx_no2_o3_basic_example (void)
{
 fsp_err_t                err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_no2_o3_data_t zmod4510_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
```

```
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
 /* Delay required time. See Table 3 in the ZMOD4510 Programming Manual. */
```

```
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_6000_MS, BSP_DELAY_UNITS_MILLISECONDS);
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
               (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
      {
 /* Error during read of sensor status or Measurement not completed due to unexpected
reset. Please reset device. */
 while (1)
      {
                  ;
      }
      }
```

```
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_DeviceErrorCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 if ((RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET == g_zmod4xxx_i2c_callback_event) ||
            (RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT ==
g_zmod4xxx_i2c_callback_event))
      {
 /* Check validness of ADC results:
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET : Unvalid ADC results due to an
unexpected reset.
  * - RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT: Unvalid ADC results due a still
running measurement while results readout.
  *//* Please reset device. */
 while (1)
      {
                ;
      }
      }
 /* Set the current temperature and humidity */
       err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl, temperature,
```

```
humidity);
      handle_error(err);
       err = RM_ZMOD4XXX_No2O3DataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4510_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4510_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

RAQ

```
void rm_zmod4xxx_raq_basic_example (void)
{
 fsp_err_t             err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t raw_data;
 rm_zmod4xxx_raq_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
```

```
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
```

```
 /* Reset ZMOD sensor (active low). Please change to the IO port connected to the
RES_N pin of the ZMOD sensor on the customer board. */
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_LOW);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
 R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_04_PIN_12, BSP_IO_LEVEL_HIGH);
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MILLISECONDS);
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if ZMOD4XXX_IRQ_ENABLE
    g_zmod4xxx_irq_callback_flag = 0;
#endif
    g_zmod4xxx_i2c_callback_flag = 0;
    err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
    handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
    {
    }
    g_zmod4xxx_i2c_callback_flag = 0;
 while (1)
    {
 do
     {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
     {
     }
     g_zmod4xxx_irq_callback_flag = 0;
#else
           err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
     handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
```

```
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_RaqDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_zmod4xxx_compensation_info_t |

| | |
|---|---|
| struct | rm_zmod4xxx_init_process_params_t |
| struct | rm_zmod4xxx_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_zmod4xxx_lib_type_t |

## Data Structure Documentation

### ◆ rm_zmod4xxx_compensation_info_t

| struct rm_zmod4xxx_compensation_info_t | | |
|---|---|---|
| ZMOD4XXX compensation info | | |
| Data Fields | | |
| rm_zmod4xxx_cfg_t const * | p_compensation_sensor_cfg | Pointer to configuration structure of compensation sensor. |
| rm_zmod4xxx_raw_data_t * | p_raw_data | Pointer to compensation sensor raw data. |

### ◆ rm_zmod4xxx_init_process_params_t

| struct rm_zmod4xxx_init_process_params_t | | |
|---|---|---|
| ZMOD4XXX initialization process block | | |
| Data Fields | | |
| volatile uint32_t | delay_ms | Delay milliseconds. |
| volatile bool | communication_finished | Communication flag for blocking. |
| volatile bool | measurement_finished | IRQ flag. |
| volatile rm_zmod4xxx_event_t | event | Callback event. |

### ◆ rm_zmod4xxx_instance_ctrl_t

| struct rm_zmod4xxx_instance_ctrl_t | |
|---|---|
| ZMOD4XXX control block | |
| **Data Fields** | |
| uint32_t | open |
| | Open flag. |
| | |
| uint8_t | buf [RM_ZMOD4XXX_MAX_I2C_BUF_SIZE] |
| | Buffer for I2C communications. |

| | |
|---|---|
| | |
| uint8_t | register_address |
| | Register address to access. |
| | |
| rm_zmod4xxx_status_params_t | status |
| | Status parameter. |
| | |
| volatile bool | dev_err_check |
| | Flag for checking device error. |
| | |
| volatile rm_zmod4xxx_event_t | event |
| | Callback event. |
| | |
| rm_zmod4xxx_init_process_params_t | init_process_params |
| | For the initialization process. |
| | |
| rm_zmod4xxx_cfg_t const * | p_cfg |
| | Pointer of configuration block. |
| | |
| rm_comms_instance_t const * | p_comms_i2c_instance |
| | Pointer of I2C Communications Middleware instance structure. |
| | |
| rm_zmod4xxx_lib_extended_cfg_t * | p_zmod4xxx_lib |
| | Pointer of ZMOD4XXX Lib extended configuration. |
| | |
| void const * | p_zmod4510_device |

| | |
|---|---|
| | Pointer of ZMOD4510 device. |
| | |
| rm_zmod4xxx_raw_data_t * | p_zmod4510_raw_data |
| | Raw data of ZMOD4510 device for calculating compensation data. |
| | |
| void const * | p_irq_instance |
| | Pointer to IRQ instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_comms_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |
| void(* | p_irq_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | IRQ callback. |
| | |

## Enumeration Type Documentation

### ◆ rm_zmod4xxx_lib_type_t

| enum rm_zmod4xxx_lib_type_t |
|---|
| ZMOD4XXX Library type |

## Function Documentation

### ◆ RM_ZMOD4XXX_Open()

fsp_err_t RM_ZMOD4XXX_Open ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_cfg_t const *const *p_cfg* )

This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the zmod4xxx and start a measurement. Implements rm_zmod4xxx_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_UNSUPPORTED | Unsupport product ID. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_ZMOD4XXX_Close()

fsp_err_t RM_ZMOD4XXX_Close ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when close the sensor. Implements rm_zmod4xxx_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully closed. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_ZMOD4XXX_MeasurementStart()

fsp_err_t RM_ZMOD4XXX_MeasurementStart ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_ZMOD4XXX_MeasurementStop()

fsp_err_t RM_ZMOD4XXX_MeasurementStop ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_ZMOD4XXX_StatusCheck()

fsp_err_t RM_ZMOD4XXX_StatusCheck ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when polling is used. It reads the status of sensor. Implements rm_zmod4xxx_api_t::statusCheck.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully started. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

### ◆ RM_ZMOD4XXX_Read()

fsp_err_t RM_ZMOD4XXX_Read ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | Communication is timeout. |
| FSP_ERR_ABORTED | Communication is aborted. |
| FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED | Measurement is not finished. |

### ◆ RM_ZMOD4XXX_Iaq1stGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Iaq1stGenDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_iaq_1st_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq1stGenDataCalculate .

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_Iaq2ndGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Iaq2ndGenDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_iaq_2nd_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_SulfurOdorDataCalculate()

fsp_err_t RM_ZMOD4XXX_SulfurOdorDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_sulfur_odor_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::sulfurOdorDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_Oaq2ndGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Oaq2ndGenDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_oaq_2nd_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq2ndGenDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_RaqDataCalculate()

fsp_err_t RM_ZMOD4XXX_RaqDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_raq_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::raqDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_RellaqDataCalculate()

fsp_err_t RM_ZMOD4XXX_RellaqDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_rel_iaq_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::rellaqDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_PbaqDataCalculate()

fsp_err_t RM_ZMOD4XXX_PbaqDataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_pbaq_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::pbaqDataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_No2O3DataCalculate()

fsp_err_t RM_ZMOD4XXX_No2O3DataCalculate ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_no2_o3_data_t *const *p_zmod4xxx_data* )

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::no2O3DataCalculate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter or library internal error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_SENSOR_IN_STABILIZATION | Module is stabilizing. |
| FSP_ERR_SENSOR_INVALID_DATA | Sensor probably damaged. Algorithm results may be incorrect. |

### ◆ RM_ZMOD4XXX_CompensationInfoSet()

fsp_err_t RM_ZMOD4XXX_CompensationInfoSet ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_compensation_info_t const *const *p_compensation_info* )

This function should be called when setting the compensation info from ZMOD4510 device for the ZMOD4410 device and when the ZMOD4510 for oxidizing gas correction option is enabled. After reading the raw data from the ZMOD4510 device, set the compensation info for the ZMOD4410 to begin calculating the compensation data.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully compensation info are set. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter or library internal error occurred. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_UNSUPPORTED | Operation mode is not supported. |

### ◆ RM_ZMOD4XXX_TemperatureAndHumiditySet()

fsp_err_t RM_ZMOD4XXX_TemperatureAndHumiditySet ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, float *temperature*, float *humidity* )

This function is valid only for OAQ_2nd_Gen and IAQ_2nd_Gen_ULP. This function should be called before DataCalculate. Humidity and temperature measurements are needed for ambient compensation. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |

### ◆ RM_ZMOD4XXX_DeviceErrorCheck()

fsp_err_t RM_ZMOD4XXX_DeviceErrorCheck ( rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function is valid only for IAQ_2nd_Gen and IAQ_2nd_Gen_ULP. This function should be called before Read and DataCalculate. Check for unexpected reset occurs or getting unvalid ADC data. Implements rm_zmod4xxx_api_t::deviceErrorCheck.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successfully results are read. |
| FSP_ERR_ASSERTION | Null pointer passed as a parameter. |
| FSP_ERR_NOT_OPEN | Module is not opened configured. |
| FSP_ERR_TIMEOUT | communication is timeout. |
| FSP_ERR_ABORTED | communication is aborted. |

## 5.2.17 Storage

Modules

### Detailed Description

Storage Modules.

### Modules

Block Media Custom Implementation (rm_block_media_user)

| | |
|---|---|
| | Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface. |
| | Block Media RAM (rm_block_media_ram)<br><br>Middleware that implements a block media interface on RAM. This module implements the Block Media Interface. |
| | Block Media SD/MMC (rm_block_media_sdmmc)<br><br>Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface. |
| | Block Media SPI Flash (rm_block_media_spi)<br><br>Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface. |
| | Block Media USB (rm_block_media_usb)<br><br>Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface. |
| | FileX I/O (rm_filex_block_media)<br><br>Middleware for the Azure RTOS FileX File System control using Block Media on RA MCUs. |
| | FileX I/O (rm_filex_levelx_nor)<br><br>Middleware for the Azure RTOS FileX File System control using LevelX NOR on RA MCUs. |
| | Flash (r_flash_hp)<br><br>Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface. |
| | Flash (r_flash_lp)<br><br>Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface. |
| | FreeRTOS+FAT Port for RA (rm_freertos_plus_fat)<br><br>Middleware for the FAT File System control on RA MCUs. |

| | |
|---|---|
| | LevelX NOR Port (rm_levelx_nor_spi)<br>Middleware for using Azure RTOS LevelX on NOR SPI memory. |
| | LittleFS on Flash (rm_littlefs_flash)<br>Middleware for the LittleFS File System control on RA MCUs. |
| | LittleFS on SPI Flash (rm_littlefs_spi_flash)<br>Middleware for the LittleFS File System control on RA MCUs using external SPI Flash. |
| | MRAM (r_mram)<br>Driver for the MRAM memory on RA MCUs. This module implements the Flash Interface. |
| | OSPI Flash (r_ospi)<br>Driver for the OSPI peripheral on RA MCUs. This module implements the SPI Flash Interface. |
| | OSPI Flash (r_ospi_b)<br>Driver for the OSPI_B peripheral on RA MCUs. This module implements the SPI Flash Interface. |
| | QSPI (r_qspi)<br>Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface. |
| | SD/MMC (r_sdhi)<br>Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface. |
| | Virtual EEPROM on Flash (rm_vee_flash)<br>Virtual EEPROM on RA MCUs. This module implements the Virtual EEPROM Interface. |

### 5.2.17.1 Block Media Custom Implementation (rm_block_media_user)

Modules » Storage

Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface.

# Overview

### Features

This module is for using Block media with user-selected media.

# Configuration

Block Media User has no output config settings.
The user is required to create the config settings etc. in the application.
The figure below is an example of the config definition when the user media in USB PMSC is RAM.

```
const rm_block_media_api_t g_rm_block_media_on_user_media =
{
    .open      = RM_BLOCK_MEDIA_RAM_Open,
    .mediaInit = RM_BLOCK_MEDIA_RAM_MediaInit,
    .read      = RM_BLOCK_MEDIA_RAM_Read,
    .write    = RM_BLOCK_MEDIA_RAM_Write,
    .erase    = RM_BLOCK_MEDIA_RAM_Erase,
    .infoGet   = RM_BLOCK_MEDIA_RAM_InfoGet,
    .statusGet = RM_BLOCK_MEDIA_RAM_StatusGet,
    .close    = RM_BLOCK_MEDIA_RAM_Close,
};
extern void r_usb_pmsc_block_media_event_callback(rm_block_media_callback_args_t *
p_args);
const rm_block_media_cfg_t g_rm_block_media0_cfg =
{.p_extend = NULL, .p_callback = r_usb_pmsc_block_media_event_callback, .p_context =
NULL, };
rm_block_media_instance_t g_rm_block_media0 =
{.p_api = &g_rm_block_media_on_user_media, .p_ctrl = NULL, .p_cfg =
&g_rm_block_media0_cfg, };
```

*Note*

*If you use block_media_user, you need to create a function that matches the media you are using.*
*In the above example, this is the function with RM_BLOCK_MEDIA_.*
*Register the created function in rm_block_media_api_t.*
*The registered rm_block_media_api_t is registered in p_api, which is a member of rm_block_media_instance_t.*

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

# Examples

### Basic Example

Please refer to USB PMSC (r_usb_pmsc) for the PMSC application given as an example.

## 5.2.17.2 Block Media RAM (rm_block_media_ram)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_RAM_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_RAM_Close (rm_block_media_ctrl_t *const p_ctrl) |

### Detailed Description

Middleware that implements a block media interface on RAM. This module implements the Block Media Interface.

# Overview

### Features

- This module is for using RAM as a media area.
- Supports using any read/write section of the address space as the media area.

*Note*

*When using an external RAM as the media area (like SDRAM or OctaRAM), it must be initialized before calling RM_BLOCK_MEDIA_RAM_Open().*
*When using PMSC driver in combination with this module, the API of this module is called from the PMSC driver.*

# Configuration

### Build Time Configurations for rm_block_media_ram

The following build time configurations are defined in fsp_cfg/rm_block_media_ram_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| RAM Media Size | Please enter the RAM media size. | 65536 | Enter a RAM media size of 20K bytes or more. |
| Memory Storage Section | Manual Entry | .bss | This is the section of memory that will be used for the backing storage. Must be a valid Linker section. ('.bss', '.sdram', '.OSPI_DEVICE_0_RAM', etc). For AC6 preface any section with '.bss' e.g. '.bss.sdram' |
| Format during mediaInit | • Enabled<br>• Disabled | Enabled | If disabled RAM must be formatted with filex_media_format before calling filex_media_open. |

### Configurations for Storage > Block Media RAM Implementation (rm_block_media_ram)

This module can be added to the Stacks tab via New Stack > Storage > Block Media RAM Implementation (rm_block_media_ram).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_block_media0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed. |

*Note*

*Specify RAM media size of 20 Kbytes or more. This module cannot be used with an MCU that cannot allocate at least 20 Kbytes of RAM as media area.*
*When using AC6 it is recommended to start section names with ".bss" (ex. ".bss.sdram") to prevent unnecessarily large binaries. This tells AC6 to create ZI sections rather than initialized sections.*

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

## Examples

### Basic Example

This is a basic example of minimal use of the RAM block media implementation in an application.

```
#define RM_BLOCK_MEDIA_RAM_BLOCK_SIZE (512U)
uint8_t g_dest[RM_BLOCK_MEDIA_RAM_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_RAM_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
rm_block_media_status_t status;
void rm_block_media_ram_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_RAM_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_RAM driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_RAM_Open(&g_rm_block_media0_ctrl,
```

```
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* When enabled, format RAM as FAT file system. */
    err = RM_BLOCK_MEDIA_RAM_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Get the RM_BLOCK_MEDIA_RAM driver status. */
    err = RM_BLOCK_MEDIA_RAM_StatusGet(&g_rm_block_media0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 if (true == status.initialized)
    {
 /* Write a block of data to block 3 of the RAM memory. */
       err = RM_BLOCK_MEDIA_RAM_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
      assert(FSP_SUCCESS == err);
 /* Read a block of data from block 3 of the RAM memory. */
       err = RM_BLOCK_MEDIA_RAM_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
      assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Close the RM_BLOCK_MEDIA_RAM driver. */
 RM_BLOCK_MEDIA_RAM_Close(&g_rm_block_media0_ctrl);
    }
}
```

## Function Documentation

### ◆ RM_BLOCK_MEDIA_RAM_Open()

fsp_err_t RM_BLOCK_MEDIA_RAM_Open ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg* )

Opens the module.

Implements rm_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is available and is now open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Module has already been opened with this instance of the control structure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_RAM_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_RAM_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the RAM media area.

Implements rm_block_media_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_RAM_Read()

fsp_err_t RM_BLOCK_MEDIA_RAM_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Reads data from RAM media.

Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_RAM_Write()

fsp_err_t RM_BLOCK_MEDIA_RAM_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Writes data to RAM media.

Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write finished successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_RAM_Erase()

fsp_err_t RM_BLOCK_MEDIA_RAM_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Erases sectors of RAM media.

Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_RAM_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_RAM_StatusGet ( rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status.

Implements rm_block_media_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_RAM_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_RAM_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information.

Implements rm_block_media_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_RAM_Close()

fsp_err_t RM_BLOCK_MEDIA_RAM_Close ( rm_block_media_ctrl_t *const *p_ctrl*)

Closes the module.

Implements rm_block_media_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## 5.2.17.3 Block Media SD/MMC (rm_block_media_sdmmc)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const |

| | |
|---|---|
| | p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const p_ctrl) |

### Detailed Description

Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.

# Overview

### Features

The SD/MMC implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from an SD card
- Callback called when card insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

# Configuration

### Build Time Configurations for rm_block_media_sdmmc

The following build time configurations are defined in driver/rm_block_media_sdmmc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Storage > Block Media SD/MMC (rm_block_media_sdmmc)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SD/MMC (rm_block_media_sdmmc). Non-secure callable guard functions can be generated for this module by

right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_block_media0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed. |

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

## Examples

### Basic Example

This is a basic example of minimal use of the SD/MMC block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE (512)
uint8_t g_dest[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void rm_block_media_sdmmc_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SDMMC driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SDMMC_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = RM_BLOCK_MEDIA_SDMMC_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

### Function Documentation

#### ◆ RM_BLOCK_MEDIA_SDMMC_Open()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Open ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg* )

Opens the module.

Implements rm_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is available and is now open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Module has already been opened with this instance of the control structure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::open

### ◆ RM_BLOCK_MEDIA_SDMMC_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements rm_block_media_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- sdmmc_api_t::mediaInit

### ◆ RM_BLOCK_MEDIA_SDMMC_Read()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements rm_block_media_api_t::read().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- sdmmc_api_t::read

### ◆ RM_BLOCK_MEDIA_SDMMC_Write()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements rm_block_media_api_t::write().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write finished successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::write

### ◆ RM_BLOCK_MEDIA_SDMMC_Erase()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Erases sectors of an SD card or eMMC device. Implements rm_block_media_api_t::erase().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::erase
- sdmmc_api_t::statusGet

### ◆ RM_BLOCK_MEDIA_SDMMC_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_CallbackSet ( rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ RM_BLOCK_MEDIA_SDMMC_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_StatusGet ( rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status. Implements rm_block_media_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_SDMMC_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information. Implements rm_block_media_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

### ◆ RM_BLOCK_MEDIA_SDMMC_Close()

| fsp_err_t RM_BLOCK_MEDIA_SDMMC_Close ( rm_block_media_ctrl_t *const *p_ctrl*) |
|---|
| Closes an open SD/MMC device. Implements rm_block_media_api_t::close(). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## 5.2.17.4 Block Media SPI Flash (rm_block_media_spi)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_SPI_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_StatusGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_Close (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_SPI_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const start_block, uint32_t const num_blocks) |

### Detailed Description

Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface.

# Overview

### Features

The SPI implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from SPI flash memory
- Provides media information such as sector size and total number of sectors.

*Note*

*By default, Block Media SPI Read, Write, and Erase are blocking operations. Non-blocking operation may be achieved by yielding control within the optional callback function.*

# Configuration

### Build Time Configurations for rm_block_media_spi

The following build time configurations are defined in driver/rm_block_media_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected, code for parameter checking is included in the build |

### Configurations for Storage > Block Media SPI Flash (rm_block_media_spi)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SPI Flash (rm_block_media_spi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Module Instance Name | Name must be a valid C symbol | g_rm_block_media0 | Module name |
| Block size (bytes) | Manual Entry | 4096 | Specify the size of a block in bytes. |
| Block count | Minimum block count is 1, maximum is defined by hardware and software design. | 8192 | Number of blocks available for use by this driver instance. |
| Base Address | Manual Entry | 0 | Base address offset (bytes) for instance memory region. |
| Callback Function | Name must be a valid C symbol | NULL | A user callback function can be |

provided. If this callback is provided, it will be called after the completion of Read, Write, and Erase operations, or anytime these functions are waiting on hardware.

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

### Limitations

Developers should be aware of the following limitations when using RM_BLOCK_MEDIA_SPI:

- Getting and setting Block Protection or Advanced Sector Protection modes is not supported.
- Addressing QSPI memory address ranges greater than 64 MB (one bank) is not supported.

## Examples

### Basic Example

This is a basic example of minimal use of the SPI block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SPI_BLOCK_SIZE (256U)
uint8_t g_dest[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_spi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SPI driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Initialize the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

### Non-Blocking Example

This is a basic example of using the optional SPI callback to implement non-blocking operation.

```
#define RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT 0x1000
void rm_block_media_spi_non_blocking_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SPI driver. This enables the card detection interrupt. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the Block Media SPI driver. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Erase a large quantity of data from SPI Flash Memory */
    err = RM_BLOCK_MEDIA_SPI_Erase(&g_rm_block_media0_ctrl, 0,
RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT);
    assert(FSP_SUCCESS == err);
```

```
}
/* The optional callback is invoked for Read, Write, and Erase operations, whenever
the operation completes or has
 * been blocked by the lower level SPI driver busy indication.
 */
void rm_block_media_spi_example_callback (rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE == p_args->event)
    {
 /* Process operation complete. */
    }
 else if (RM_BLOCK_MEDIA_EVENT_POLL_STATUS == p_args->event)
    {
 rm_block_media_status_t status;
 rm_block_media_ctrl_t * p_ctrl = (rm_block_media_ctrl_t *) p_args->p_context;
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_StatusGet(p_ctrl, &status);
      assert(FSP_SUCCESS == err);
 if (true == status.busy)
      {
 /* Run waiting tasks */
            vTaskSuspend(xTaskGetCurrentTaskHandle());
      }
    }
 else
    {
       assert(RM_BLOCK_MEDIA_EVENT_ERROR == p_args->event);
 /* Process Read, Write, or Erase error. */
    }
}
```

### Function Documentation

### ◆ RM_BLOCK_MEDIA_SPI_Open()

fsp_err_t RM_BLOCK_MEDIA_SPI_Open ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg* )

Parameter checking and Acquires mutex, then handles driver initialization at the HAL SPI layer and marking the open flag in control block.

Implements rm_block_media_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Block media for SPI framework is successfully opened. |
| FSP_ERR_ASSERTION | One of the input parameters or their data references may be null. |
| FSP_ERR_ALREADY_OPEN | The channel specified has already been opened. See HAL driver for other possible causes. |

**Returns**

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::open

### ◆ RM_BLOCK_MEDIA_SPI_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information.

Implements rm_block_media_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

### ◆ RM_BLOCK_MEDIA_SPI_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SPI_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the Block Media SPI Flash device.

Implements rm_block_media_api_t::mediaInit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_SPI_Read()

fsp_err_t RM_BLOCK_MEDIA_SPI_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *num_blocks* )

Reads a number of blocks from spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::read.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI data read successfully |
| FSP_ERR_ASSERTION | p_ctrl or p_dest is NULL, or num_blocks is zero |
| FSP_ERR_NOT_OPEN | Block Media SPI module is not yet open |
| FSP_ERR_INVALID_ADDRESS | Invalid address range for read operation |
| FSP_ERR_NOT_INITIALIZED | Block Media SPI module is not yet initialized |

### ◆ RM_BLOCK_MEDIA_SPI_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_StatusGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status.

Implements rm_block_media_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |

**Returns**

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls
- spi_flash_api_t::statusGet

### ◆ RM_BLOCK_MEDIA_SPI_Write()

fsp_err_t RM_BLOCK_MEDIA_SPI_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *num_blocks* )

Writes provided data to a number of blocks of spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash write finished successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_src is NULL. Or num_blocks is zero. |
| FSP_ERR_NOT_OPEN | Block media SPI Framework module is not yet initialized. |
| FSP_ERR_INVALID_ADDRESS | Invalid address range |
| FSP_ERR_NOT_INITIALIZED | Block Media SPI module is not yet initialized |

**Returns**

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls
- spi_flash_api_t::write

### ◆ RM_BLOCK_MEDIA_SPI_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SPI_CallbackSet ( rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. API not supported.

Implements rm_block_media_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by RM_BLOCK_MEDIA_SPI. |

### ◆ RM_BLOCK_MEDIA_SPI_Close()

fsp_err_t RM_BLOCK_MEDIA_SPI_Close ( rm_block_media_ctrl_t *const *p_ctrl*)

Closes the Block Media SPI device. Implements rm_block_media_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | One of the following parameters may be null: p_ctrl. |
| FSP_ERR_NOT_OPEN | Block media SPI Framework module is not yet initialized. |

**Returns**

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::close

### ◆ RM_BLOCK_MEDIA_SPI_Erase()

fsp_err_t RM_BLOCK_MEDIA_SPI_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *start_block*, uint32_t const *num_blocks* )

This function erases blocks of the SPI device. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::erase.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |
| FSP_ERR_INVALID_ADDRESS | Invalid address range |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- spi_flash_api_t::erase
- spi_flash_api_t::statusGet

## 5.2.17.5 Block Media USB (rm_block_media_usb)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const p_ctrl) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_CallbackSet (rm_block_media_ctrl_t *const |

| | |
|---|---|
| | p_api_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| fsp_err_t | RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const p_ctrl) |

### Detailed Description

Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.

# Overview

### Features

The USB implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from a USB mass storage device
- Callback called when device insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

# Configuration

### Build Time Configurations for rm_block_media_usb

The following build time configurations are defined in driver/rm_block_media_usb_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Storage > Block Media USB (rm_block_media_usb)

This module can be added to the Stacks tab via New Stack > Storage > Block Media USB (rm_block_media_usb). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_block_media0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be |

| | | | |
|---|---|---|---|
| | | | provided. If this callback function is provided, it will be called when a device is attached or removed. |
| Pointer to user context | Name must be a valid C symbol | NULL | A user context can be provided. If this context is provided, it will be passed to callback function when a device is attached or removed. |

*Note*

*RM_BLOCK_MEDIA_USB_MediaInit function must be called after receiving the insert event notification.*

### Clock Configuration

This module has no required clock configurations.

### Pin Configuration

This module does not use I/O pins.

## Examples

### Basic Example

This is a basic example of minimal use of the USB mass storage block media implementation in an application.

```
#define RM_BLOCK_MEDIA_USB_BLOCK_SIZE (512)
volatile bool g_usb_inserted = false;
uint8_t       g_dest[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t       g_src[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_usb_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_USB_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_USB driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_USB_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!g_usb_inserted)
    {
 /* Wait for a card insertion interrupt. */
    }
 /* Initialize the mass storage device. This should not be done until the device is
plugged in and initialized. */
    err = RM_BLOCK_MEDIA_USB_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

### Device Insertion

This is an example of using a callback to determine when a mass storage device is plugged in and enumerated.

```
/* The callback is called when a media insertion event occurs. */
void rm_block_media_usb_media_insertion_example_callback
(rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED == p_args->event)
    {
       g_usb_inserted = true;
    }
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED == p_args->event)
    {
       g_usb_inserted = false;
    }
```

```
}
```

## Function Documentation

### ◆ RM_BLOCK_MEDIA_USB_Open()

fsp_err_t RM_BLOCK_MEDIA_USB_Open ( rm_block_media_ctrl_t *const *p_ctrl*,
rm_block_media_cfg_t const *const *p_cfg* )

Opens the module.

Implements rm_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is available and is now open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_ALREADY_OPEN | Module has already been opened with this instance of the control structure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_USB_MediaInit ( rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the USB device.

Implements rm_block_media_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ RM_BLOCK_MEDIA_USB_Read()

fsp_err_t RM_BLOCK_MEDIA_USB_Read ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Reads data from an USB device. Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |
| FSP_ERR_USB_FAILED | The message could not received completed successfully. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_Write()

fsp_err_t RM_BLOCK_MEDIA_USB_Write ( rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Writes data to an USB device. Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write finished successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |
| FSP_ERR_USB_FAILED | The message could not received completed successfully. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_Erase()

fsp_err_t RM_BLOCK_MEDIA_USB_Erase ( rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks* )

Erases sectors of an USB device. Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_USB_CallbackSet ( rm_block_media_ctrl_t *const *p_api_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

***Note***

*This function is currently unsupported for Block Media over USB.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | CallbackSet is not currently supported for Block Media over USB. |

### ◆ RM_BLOCK_MEDIA_USB_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_USB_StatusGet ( rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status* )

Provides driver status. Implements rm_block_media_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_BLOCK_MEDIA_USB_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_USB_InfoGet ( rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info* )

Retrieves module information. Implements rm_block_media_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_INITIALIZED | Module has not been initialized. |

### ◆ RM_BLOCK_MEDIA_USB_Close()

fsp_err_t RM_BLOCK_MEDIA_USB_Close ( rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open USB device. Implements rm_block_media_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### 5.2.17.6 FileX I/O (rm_filex_block_media)

Modules » Storage

**Functions**

| | |
|---|---|
| fsp_err_t | RM_FILEX_BLOCK_MEDIA_Open (rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg) |
| fsp_err_t | RM_FILEX_BLOCK_MEDIA_Close (rm_filex_block_media_ctrl_t *const p_ctrl) |
| void | RM_FILEX_BLOCK_MEDIA_BlockDriver (FX_MEDIA *p_fx_media)<br>Access Block Media device functions open, close, read, write and control. More... |

**Detailed Description**

Middleware for the Azure RTOS FileX File System control using Block Media on RA MCUs.

# Overview

This module provides the hardware port layer for FileX file system. After initializing this module, refer to the FileX API reference to use the file system: https://docs.microsoft.com/en-us/azure/rtos/filex/

### Features

The FileX Block Media module supports the following features:

- Callbacks for insertion and removal for removable devices.
- ThreadX is typically required for FileX. To use FileX without ThreadX FX_STANDALONE_ENABLE must be defined.
- Unless FX_SINGLE_THREAD or FX_STANDALONE_ENABLE are defined, all FileX operations are thread safe.

# Configuration

### Build Time Configurations for rm_filex_block_media

The following build time configurations are defined in fsp_cfg/middleware/rm_filex_block_media_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Selects if code for parameter checking is to be included in the build. |

### Configurations for Storage > FileX I/O (rm_filex_block_media)

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_filex_block_media_0 | Module name. |
| Callback | Name must be a valid C symbol | g_rm_filex_block_media_0_callback | A user callback function can be provided. If this callback function is provided, it will be called when media is inserted or removed. It will also be called during operations by the lower level block media as a way for the user to provide their desired waiting functionality. |
| Partition Number | • 0<br>• 1<br>• 2<br>• 3 | 0 | The partition to use for partitioned media. This partition will only be used if a Master Boot Record with partition table exists at block 0 of the media, otherwise the FileX FAT boot record should exist or be formatted to block 0. |

### Build Time Configurations for fx

The following build time configurations are defined in fsp_cfg/azure/fx/fx_user.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Common | | | |
| Max Long Name Len | Value must be an integer greater than or equal to 13 and less than or equal to 256, or empty | | Specifies the maximum file name size for FileX. If left blank the default value is 256. Legal values range between 13 and 256. |
| Max Last Name Len | Value must be an integer greater than or equal to 13 and less than or equal to 256, or empty | | This value defines the maximum file name length, which includes full path name. If left blank the default value is 256. Legal values range between 13 and 256. |

| | | | |
|---|---|---|---|
| Max Sector Cache | Value must be an integer greater than 0 and power of 2 or empty | | Specifies the maximum number of logical sectors that can be cached by FileX. The actual number of sectors that can be cached is lesser of this constant and how many sectors can fit in the amount of memory supplied at fx_media_open. The default value if left blank is 256. All values must be a power of 2. |
| Fat Map Size | Value must be an integer greater than 0 or empty | | Specifies the number of sectors that can be represented in the FAT update map. The default value if left blank is 256. Larger values help reduce unneeded updates of secondary FAT sectors. |
| Max Fat Cache | Value must be an integer greater than 0 and power of 2 or empty | | Specifies the number of entries in the internal FAT cache. The default value if left blank is 16. All values must be a power of 2. |
| Threading | | | |
| Update Rate (Seconds) | Value must be an integer greater than 0 or empty | | Specifies rate at which system time in FileX is adjusted. Default value if left blank is 10, specifying that the FileX system time is updated every 10 seconds. |
| No Timer | • Enabled<br>• Disabled (default) | Disabled (default) | Eliminates the ThreadX timer setup to update the FileX system time and date. Doing so causes default time and date to be placed on all file operations. |
| Single Thread | • Enabled<br>• Disabled (default) | Disabled (default) | Eliminates ThreadX protection logic from the FileX source. It should be used if FileX is being used only from |

| | | | |
|---|---|---|---|
| | | | one thread. |
| Standalone | • Enabled<br>• Disabled (default) | Disabled (default) | Enables FileX to be used in standalone mode (without Azure RTOS). |
| Extra Features | | | |
| Don't Update Open Files | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX does not update already opened files. |
| Media Search Cache | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, the file search cache optimization is disabled. |
| Direct Data Read Cache Fill | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, the direct read sector update of cache is disabled. |
| Media Statistics | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, gathering of media statistics is disabled. |
| Single Open Legacy | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, legacy single open logic for the same file is enabled. |
| Rename Path Inherit | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, renaming inherits path information. |
| No Local Path | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, removes local path logic from FileX, resulting in smaller code size. |
| 64-bit LBA | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, 64-bits sector addresses are used in I/O driver. |
| Cache | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables the cache, default is enabled. |
| File Close | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables file close, default is enabled. |
| Fast Close | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables fast open, default is enabled. |
| Force Memory Operation | • Enabled (default) | Enabled (default) | Enables or disables force memory |

| | | | |
|---|---|---|---|
| | • Disabled | | operation, default is enabled. |
| Build Options | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables build options, default is enabled. |
| One Line Function | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables one line function, default is enabled. |
| FAT Entry Refresh | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables FAT entry refresh, default is enabled. |
| Consecutive Detect | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables consecutive detect, default is enabled. |
| Enable exFAT | • Enabled<br>• Disabled (default) | Disabled (default) | Enables exFAT support in FileX. |
| Fault Tolerant | | | |
| Fault Tolerant Service | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, enables the FileX Fault Tolerant Module. Enabling Fault Tolerant automatically defines the symbol FX_FAULT_TOLERANT and FX_FAULT_TOLERANT_DATA. |
| Fault Tolerant Data | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX immediately passes all file data write requests to the media's driver. This potentially decreases performance, but helps limit lost file data. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately. |
| Fault Tolerant | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX immediately passes write requests of all system sectors (boot, FAT, and directory sectors) to the media's driver. This potentially decreases performance, but helps |

| | | | |
|---|---|---|---|
| | | | limit corruption to lost clusters. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately. |
| Fault Tolerant Boot Index | Value must be an integer greater than or equal to 116 and less than or equal to 119 | | Defines byte offset in the boot sector where the cluster for the fault tolerant log is. By default if left blank this value is 116. This field takes 4 bytes. Bytes 116 through 119 are chosen because they are marked as reserved by FAT 12/16/32/exFAT specification. |
| Error Checking | • Enabled (default)<br>• Disabled | Enabled (default) | |

**Configurations for Storage > Azure RTOS FileX on Block Media**

This module can be added to the Stacks tab via New Stack > Storage > Azure RTOS FileX on Block Media.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_fx_media0 | Symbol used for media_ptr parameter in FileX APIs |
| Volume Name | Name must be a maximum of 11 characters | Volume 1 | Volume name string, which is a maximum of 11 characters. |
| Number of FATs | Number of FATs must be an integer greater than 0 | 1 | Number of FATs in the media. The minimal value is 1 for the primary FAT. Values greater than 1 result in additional FAT copies being maintained at run-time. |
| Directory Entries | Number of Directory Entries must be an integer greater than 0 | 256 | Number of directory entries in the root directory. |
| Hidden Sectors | Number of Hidden Sectors must be an | 0 | Number of sectors hidden before this |

| | | | |
|---|---|---|---|
| | integer | | media's boot sector. If using media formatted with multiple partitions this number should correspond to the starting block number for the desired partition. |
| Total Sectors | Total Sectors must be an integer greater than 0 | 65536 | Total number of sectors in the media. When using a Renesas provided block media implementation, total sectors can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info. |
| Bytes per Sector | Bytes per Sector must be multiple of 32 | 512 | Number of bytes per sector, which is typically 512. FileX requires this to be a multiple of 32. When using a Renesas provided block media implementation, bytes per sector can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info. |
| Sectors per Cluster | Sectors per Cluster must be an integer greater than 0 | 1 | Number of sectors in each cluster. The cluster is the minimum allocation unit in a FAT file system. |
| Volume Serial Number (exFAT only) | Volume Serial Number must be an integer greater than 0 | 12345 | Serial number to be used for this volume. exFAT only. |
| Boundary Unit (exFAT only) | Boundary unit must be an integer greater than 0 | 128 | Physical data area alignment size, in number of sectors. exFAT only. |
| Working media memory size | Memory size must be an integer greater than or equal to the size of one sector | 512 | Memory allocated for file system. Memory size must be an integer greater than or equal |

to the size of one sector.

# Usage Notes

### Pending during Read/Write

The FileX Block Media driver provides a number of events in the user callback to handle waiting or pending while it is doing blocking operations. The events received in the callback will differ depending on the lower level block media driver in use.

If the lower level block media driver is rm_block_media_spi (SPI blocks on read/write operations):

- The user will receive RM_BLOCK_MEDIA_EVENT_POLL_STATUS in the user callback while the lower level driver is polling for the read/write operation to be complete. The user can choose to do a thread sleep or software delay upon receiving this event in the callback.
- Once the operation is complete no other callbacks will be received.

If the lower level block media driver is rm_block_media_sdmmc (SDMMC is interrupt based, the FileX Block Media driver will still block while waiting for interrupts from SDMMC):

- The user will receive RM_BLOCK_MEDIA_EVENT_WAIT in the user callback when the FileX Block Media driver begins waiting for an interrupt event from SDMMC. This is sent from a thread context. The user can choose to pend on a semaphore, sleep the thread, or do a software delay upon receiving this event in the callback. The FileX Block Media driver thread will block until an interrupt is received.
- Once an SDMMC interrupt is received the user will receive RM_BLOCK_MEDIA_EVENT_WAIT_END in the user callback. This is sent from an interrupt context. The user can choose to give a semaphore on this event or do nothing.
- If SDMMC is busy on a long erase after receiving the interrupt the FileX Block Media driver will send RM_BLOCK_MEDIA_EVENT_POLL_STATUS to the user callback and proceed to do a blocking poll on SDMMC status. The user can choose to do a thread sleep or software delay upon receiving this event in the callback. This event will not be received by the user on typical operations by FileX.

### Partitioned Media

When using fx_format to format a partition the number of hidden sectors should match the starting block number of the partition and the total number of sectors should be equal to the number of sectors in the partition.

### Unused User Callback Events

Certain events are defined in rm_block_media_event_t but not returned by the FileX Block Media user callback:

- RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE: This event is handled internally and operation success is indicated by FileX API calls returning FX_SUCCESS.
- RM_BLOCK_MEDIA_EVENT_ERROR: This event is handled internally and operation failure will be indicated by an error return code from FileX API calls.

### Erasing Flash Memory Prior to Usage

The area of the flash memory being used for the FileX instance should be erased using the lower

level flash API prior to usage. Otherwise, FileX open operation may fail when fx_media_open( ) is called.

# Examples

### Basic Example

This is a basic example of FileX Block Media in an application.

```
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_PARTITION_NUMBER (0)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_BLOCK_SIZE (512)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES (128)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS (0)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS (1073741824)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE (512)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK (1)
extern rm_filex_block_media_instance_t g_filex_block_media0;
extern rm_filex_block_media_instance_ctrl_t g_filex_block_media0_ctrl;
extern rm_filex_block_media_cfg_t g_filex_block_media0_cfg;
extern FX_MEDIA g_fx_media0;
extern uint8_t g_fx_media0_memory[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BLOCK_SIZE];
extern uint8_t g_file_data[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_filex_block_media_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize FileX */
    fx_system_initialize();
```

```
 /* Open the media. If the media is removable, it must be inserted before calling
  * fx_media_open. This assumes the disk is already partitioned and formatted. */
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_BLOCK_MEDIA_BlockDriver,
                              &g_filex_block_media0,
                              g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
 /* Create a file */
    fx_err = fx_file_create(&g_fx_media0, RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME);
    handle_fx_error(fx_err);
 /* Open source file for writing. */
    FX_FILE sourceFile;
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME, FX_OPEN_FOR_WRITE);
    handle_fx_error(fx_err);
 /* Write file data. */
    fx_err = fx_file_write(&sourceFile, g_file_data, sizeof(g_file_data));
    handle_fx_error(fx_err);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Open the source file in read mode. */
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME, FX_OPEN_FOR_READ);
    handle_fx_error(fx_err);
 /* Read file data. */
    ULONG actual_size_read;
    fx_err = fx_file_read(&sourceFile, g_read_buffer, sizeof(g_file_data),
&actual_size_read);
    handle_fx_error(fx_err);
    assert(sizeof(g_file_data) == actual_size_read);
 /* Close the file. */
```

```
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
 /* Close the Media */
    fx_err = fx_media_close(&g_fx_media0);
    handle_fx_error(fx_err);
}
```

### Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_filex_block_media_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the media */
    UINT fx_err =
fx_media_format(&g_fx_media0,                                    // Pointer to
FileX media control block.
 RM_FILEX_BLOCK_MEDIA_BlockDriver,                  // Driver entry
                                &g_filex_block_media0, // Pointer to Block Media
Driver
                                g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                 // Volume Name
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS,
// Number of FATs
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS,
```

```
// Hidden sectors

RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS,         // Total sectors
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS,
// Heads
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

### Callback Pend Example

This shows how to use the I/O driver callback with ThreadX in order to wait/pend for operations to complete.

```
TX_SEMAPHORE g_operation_wait_semaphore;
/* Callback called by FileX block media I/O driver needs to pend on operation. */
void rm_filex_block_media_test_callback_pend (rm_filex_block_media_callback_args_t *
p_args)
{
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_WAIT)
    {
 /* Interrupt has not happened for operation, get semaphore to wait for it. This will
be called from the FileX I/O driver thread. */
       tx_semaphore_get(&g_operation_wait_semaphore, TX_WAIT_FOREVER);
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_WAIT_END)
    {
 /* Interrupt has occurred for operation, post semaphore so that wait will end. This
will be called from an interrupt context. */
       tx_semaphore_put(&g_operation_wait_semaphore);
```

```
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_POLL_STATUS)
    {
 /* Interrupt has been received from block media device but operation is still
ongoing. The FileX I/O driver will wait on the driver busy status.
  * This event can be used to put the thread to sleep while waiting. This will be
called from the FileX I/O driver thread. */
       tx_thread_sleep(1);
    }
}
void rm_filex_block_media_callback_pend_example (void)
{
 /* Create semaphore for driver use */
    tx_semaphore_create(&g_operation_wait_semaphore, "operation_wait_semaphore", 0);
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the media */
    UINT fx_err =
fx_media_format(&g_fx_media0,                                    // Pointer to
FileX media control block.
 RM_FILEX_BLOCK_MEDIA_BlockDriver,                  // Driver entry
                                &g_filex_block_media0, // Pointer to Block Media
Driver
                                g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                 // Volume Name
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS,
// Number of FATs
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
                                 RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS,
```

```
// Hidden sectors

RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS,         // Total sectors
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS,
// Heads
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

### Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
volatile uint32_t g_rm_filex_block_media_insertion_events = 0;
volatile uint32_t g_rm_filex_block_media_removal_events = 0;
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_filex_block_media_test_callback (rm_filex_block_media_callback_args_t *
p_args)
{
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED)
    {
       g_rm_filex_block_media_insertion_events++;
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED)
    {
       g_rm_filex_block_media_removal_events++;
    }
}
void rm_filex_block_media_media_insertion_example (void)
{
```

```
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for media insertion. */
 while (0U == g_rm_filex_block_media_insertion_events)
    {
 /* Wait for media insertion. */
    }
 /* Open the media. If the media is removable, it must be inserted before calling
  * fx_media_open. This assumes the disk is already partitioned and formatted. */
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_BLOCK_MEDIA_BlockDriver,
                             &g_filex_block_media0,
                             g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
}
```

### Using FileX with Custom Block Media Implementations

When using a Custom Block Media implementation with rm_filex_block_media the custom implementation must call rm_filex_block_media_memory_callback upon the completion of a read/write operation. This callback should be called with an event of RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE and p_context of rm_filex_block_media_instance_ctrl_t *. The following example shows how this should be done in the context of a demo RAM block media read function.

```
#define EXAMPLE_BLOCK_MEDIA_RAM_START_ADDR (0x20004AFE)
#define EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES (512)
/* Example implementation of rm_block_media_api_t::read(), user should define custom
block media RAM implementation. */
fsp_err_t RM_BLOCK_MEDIA_CUSTOM_RAM_Read (rm_block_media_ctrl_t * const p_ctrl,
                                          uint8_t * const
               p_dest_address,
```

```
                                       uint32_t const
block_address,
                                       uint32_t const                num_blocks)
{
 FSP_PARAMETER_NOT_USED(p_ctrl);
    memcpy(p_dest_address,
          (void *) (EXAMPLE_BLOCK_MEDIA_RAM_START_ADDR + (block_address *
EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES)),
          (EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES * num_blocks));
 /* Notify FileX port of operation complete through calling the callback, this is
required for custom block media/FileX port integration */
 rm_block_media_callback_args_t args;
    args.event    = RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE;
    args.p_context = &g_filex_block_media0_ctrl;
    rm_filex_block_media_memory_callback(&args);
 return FSP_SUCCESS;
}
```

## Data Structures

| | |
|---|---|
| struct | rm_filex_block_media_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_filex_block_media_instance_ctrl_t

| struct rm_filex_block_media_instance_ctrl_t |
|---|
| Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. FileX block media private control block. DO NOT MODIFY. Initialization occurs when RM_FILEX_BLOCK_MEDIA_Open is called. |

## Function Documentation

### ◆ RM_FILEX_BLOCK_MEDIA_Open()

fsp_err_t RM_FILEX_BLOCK_MEDIA_Open ( rm_filex_block_media_ctrl_t *const *p_ctrl*, rm_filex_block_media_cfg_t const *const *p_cfg* )

The file system relies on the media to be formatted prior to creating directories and files The sector size and sector count will change depending on the media type and size.

The File Allocation Table (FAT) starts after the reserved sectors in the media. The FAT area is basically an array of 12-bit, 16-bit, or 32-bit entries that determine if that cluster is allocated or part of a chain of clusters comprising a subdirectory or a file. The size of each FAT entry is determined by the number of clusters that need to be represented. If the number of clusters (derived from the total sectors divided by the sectors per cluster) is less than 4,086, 12-bit FAT entries are used. If the total number of clusters is greater than 4,086 and less than or equal to 65,525, 16-bit FAT entries are used. Otherwise, if the total number of clusters is greater than 65,525, 32-bit FAT entries are used. Initializes callback and configuration for FileX Block Media interface. Call this before calling any FileX functions.

Implements rm_filex_block_media_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_FILEX_BLOCK_MEDIA_Close()

fsp_err_t RM_FILEX_BLOCK_MEDIA_Close ( rm_filex_block_media_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_filex_block_media_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ RM_FILEX_BLOCK_MEDIA_BlockDriver()

void RM_FILEX_BLOCK_MEDIA_BlockDriver ( FX_MEDIA * *p_fx_media*)

Access Block Media device functions open, close, read, write and control.

The RM_FILEX_BLOCK_MEDIA_BlockDriver function is called from the FileX file system driver and issues requests to a Block Media device through the FSP Block Media Interface. Uses block media driver for accesses.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_fx_media | FileX media control block. All information about each open media device are maintained in the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide. |

**Return values**

| | |
|---|---|
| None | |

**Returns**

Nothing, but updates FileX media control block.

## 5.2.17.7 FileX I/O (rm_filex_levelx_nor)

Modules » Storage

### Functions

| | |
|---|---|
| void | RM_FILEX_LEVELX_NOR_DeviceDriver (FX_MEDIA *p_fx_media)<br>Access LevelX NOR device functions open, close, read, write and control. More... |

### Detailed Description

Middleware for the Azure RTOS FileX File System control using LevelX NOR on RA MCUs.

# Overview

This module provides the hardware port layer for FileX file system. After initializing this module, refer to the FileX API reference to use the file system: https://docs.microsoft.com/en-us/azure/rtos/filex/

### Features

The FileX LevelX NOR module supports the following features:

- ThreadX is typically required for FileX. To use FileX without ThreadX FX_STANDALONE_ENABLE must be defined.
- Unless FX_SINGLE_THREAD or FX_STANDALONE_ENABLE are defined, all FileX operations are thread safe.

## Configuration

### Build Time Configurations for rm_filex_levelx_nor

The following build time configurations are defined in fsp_cfg/middleware/rm_filex_levelx_nor_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Selects if code for parameter checking is to be included in the build. |

### Configurations for Storage > FileX I/O (rm_filex_levelx_nor)

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_filex_levelx_nor0 | Module name. |
| Callback | Name must be a valid C symbol | g_rm_filex_levelx_nor_0_callback | A user callback function can be provided. If this callback function is provided, it will be called during operations by the lower level block media as a way for the user to provide their desired waiting functionality. |
| LevelX NOR Name (String) | Manual Entry | g_rm_filex_levelx_nor_0 | String name to be input into LevelX API. |

### Build Time Configurations for fx

The following build time configurations are defined in fsp_cfg/azure/fx/fx_user.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Common | | | |
| Max Long Name Len | Value must be an | | Specifies the maximum |

| | | |
|---|---|---|
| | integer greater than or equal to 13 and less than or equal to 256, or empty | file name size for FileX. If left blank the default value is 256. Legal values range between 13 and 256. |
| Max Last Name Len | Value must be an integer greater than or equal to 13 and less than or equal to 256, or empty | This value defines the maximum file name length, which includes full path name. If left blank the default value is 256. Legal values range between 13 and 256. |
| Max Sector Cache | Value must be an integer greater than 0 and power of 2 or empty | Specifies the maximum number of logical sectors that can be cached by FileX. The actual number of sectors that can be cached is lesser of this constant and how many sectors can fit in the amount of memory supplied at fx_media_open. The default value if left blank is 256. All values must be a power of 2. |
| Fat Map Size | Value must be an integer greater than 0 or empty | Specifies the number of sectors that can be represented in the FAT update map. The default value if left blank is 256. Larger values help reduce unneeded updates of secondary FAT sectors. |
| Max Fat Cache | Value must be an integer greater than 0 and power of 2 or empty | Specifies the number of entries in the internal FAT cache. The default value if left blank is 16. All values must be a power of 2. |
| Threading | | |
| Update Rate (Seconds) | Value must be an integer greater than 0 or empty | Specifies rate at which system time in FileX is adjusted. Default value if left blank is 10, specifying that the FileX system time is updated every 10 |

| | | | |
|---|---|---|---|
| | | | seconds. |
| No Timer | • Enabled<br>• Disabled (default) | Disabled (default) | Eliminates the ThreadX timer setup to update the FileX system time and date. Doing so causes default time and date to be placed on all file operations. |
| Single Thread | • Enabled<br>• Disabled (default) | Disabled (default) | Eliminates ThreadX protection logic from the FileX source. It should be used if FileX is being used only from one thread. |
| Standalone | • Enabled<br>• Disabled (default) | Disabled (default) | Enables FileX to be used in standalone mode (without Azure RTOS). |
| Extra Features | | | |
| Don't Update Open Files | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX does not update already opened files. |
| Media Search Cache | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, the file search cache optimization is disabled. |
| Direct Data Read Cache Fill | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, the direct read sector update of cache is disabled. |
| Media Statistics | • Enabled (default)<br>• Disabled | Enabled (default) | When disabled, gathering of media statistics is disabled. |
| Single Open Legacy | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, legacy single open logic for the same file is enabled. |
| Rename Path Inherit | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, renaming inherits path information. |
| No Local Path | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, removes local path logic from FileX, resulting in smaller code size. |
| 64-bit LBA | • Enabled<br>• Disabled | Disabled (default) | When enabled, 64-bits sector addresses are |

| | | | |
|---|---|---|---|
| | (default) | | used in I/O driver. |
| Cache | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables the cache, default is enabled. |
| File Close | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables file close, default is enabled. |
| Fast Close | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables fast open, default is enabled. |
| Force Memory Operation | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables force memory operation, default is enabled. |
| Build Options | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables build options, default is enabled. |
| One Line Function | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables one line function, default is enabled. |
| FAT Entry Refresh | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables FAT entry refresh, default is enabled. |
| Consecutive Detect | • Enabled (default)<br>• Disabled | Enabled (default) | Enables or disables consecutive detect, default is enabled. |
| Enable exFAT | • Enabled<br>• Disabled (default) | Disabled (default) | Enables exFAT support in FileX. |
| Fault Tolerant | | | |
| Fault Tolerant Service | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, enables the FileX Fault Tolerant Module. Enabling Fault Tolerant automatically defines the symbol FX_FAULT_TOLERANT and FX_FAULT_TOLERANT_DATA. |
| Fault Tolerant Data | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX immediately passes all file data write requests to the media's driver. This potentially decreases performance, but helps limit lost file data. Note that enabling this |

| | | | |
|---|---|---|---|
| | | | feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately. |
| Fault Tolerant | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, FileX immediately passes write requests of all system sectors (boot, FAT, and directory sectors) to the media's driver. This potentially decreases performance, but helps limit corruption to lost clusters. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately. |
| Fault Tolerant Boot Index | Value must be an integer greater than or equal to 116 and less than or equal to 119 | | Defines byte offset in the boot sector where the cluster for the fault tolerant log is. By default if left blank this value is 116. This field takes 4 bytes. Bytes 116 through 119 are chosen because they are marked as reserved by FAT 12/16/32/exFAT specification. |
| Error Checking | • Enabled (default)<br>• Disabled | Enabled (default) | |

**Configurations for Storage > Azure RTOS FileX on LevelX NOR**

This module can be added to the Stacks tab via New Stack > Storage > Azure RTOS FileX on LevelX NOR.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_fx_media0 | Symbol used for media_ptr parameter in FileX APIs |
| Volume Name | Name must be a maximum of 11 characters | Volume 1 | Volume name string, which is a maximum of 11 characters. |

| | | | |
|---|---|---|---|
| Number of FATs | Number of FATs must be an integer greater than 0 | 1 | Number of FATs in the media. The minimal value is 1 for the primary FAT. Values greater than 1 result in additional FAT copies being maintained at run-time. |
| Directory Entries | Number of Directory Entries must be an integer greater than 0 | 256 | Number of directory entries in the root directory. |
| Hidden Sectors | Number of Hidden Sectors must be an integer | 0 | Number of sectors hidden before this media's boot sector. If using media formatted with multiple partitions this number should correspond to the starting block number for the desired partition. |
| Total Sectors | Total Sectors must be an integer greater than 0 | 57337 | Total number of sectors in the media. When using a Renesas provided block media implementation, total sectors can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info. |
| Sectors per Cluster | Sectors per Cluster must be an integer greater than 0 | 1 | Number of sectors in each cluster. The cluster is the minimum allocation unit in a FAT file system. |
| Volume Serial Number (exFAT only) | Volume Serial Number must be an integer greater than 0 | 12345 | Serial number to be used for this volume. exFAT only. |
| Boundary Unit (exFAT only) | Boundary unit must be an integer greater than 0 | 128 | Physical data area alignment size, in number of sectors. exFAT only. |
| Working media memory size | Memory size must be an integer greater than or equal to the size of one sector | 512 | Memory allocated for file system. Memory size must be an integer greater than or equal to the size of one sector. |

# Usage Notes

### Pending during Write/Erase

If the underlying LevelX NOR driver performs a blocking operation that requires waiting to complete (such as a long write/erase on NOR SPI), a callback can be provided to provide a way to wait with an OS-specific thread wait. This callback will also pass up block erase events.

### Partitioned Media

Partitioned media is not supported directly by the FileX LevelX NOR port.

# Examples

### Basic Example

This is a basic example of FileX Block Media in an application.

```
#define RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_PARTITION_NUMBER (0)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_BLOCK_SIZE (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES (128)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS (0)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK (1)
extern rm_filex_levelx_nor_instance_t g_filex_levelx_nor0;
extern rm_filex_levelx_nor_instance_ctrl_t g_filex_levelx_nor0_ctrl;
extern rm_filex_levelx_nor_cfg_t g_filex_levelx_nor0_cfg;
extern FX_MEDIA g_fx_media0;
extern uint8_t g_fx_media0_memory[RM_FILEX_LEVELX_NOR_EXAMPLE_BLOCK_SIZE];
extern uint8_t g_file_data[RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_filex_levelx_nor_example (void)
{
 /* Initialize FileX */
```

```
    fx_system_initialize();
 /* Initialize LevelX */
    lx_nor_flash_initialize();
 /* Open the media. This assumes the flash is already formatted. */
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_LEVELX_NOR_DeviceDriver,
                             &g_filex_levelx_nor0,
                             g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
 /* Create a file */
    fx_err = fx_file_create(&g_fx_media0, RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME);
    handle_fx_error(fx_err);
 /* Open source file for writing. */
    FX_FILE sourceFile;
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME, FX_OPEN_FOR_WRITE);
    handle_fx_error(fx_err);
 /* Write file data. */
    fx_err = fx_file_write(&sourceFile, g_file_data, sizeof(g_file_data));
    handle_fx_error(fx_err);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Open the source file in read mode. */
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME, FX_OPEN_FOR_READ);
    handle_fx_error(fx_err);
 /* Read file data. */
    ULONG actual_size_read;
    fx_err = fx_file_read(&sourceFile, g_read_buffer, sizeof(g_file_data),
&actual_size_read);
    handle_fx_error(fx_err);
```

```
    assert(sizeof(g_file_data) == actual_size_read);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
 /* Close the Media */
    fx_err = fx_media_close(&g_fx_media0);
    handle_fx_error(fx_err);
}
```

### Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
extern rm_levelx_nor_spi_cfg_t g_levelx_nor_spi0_cfg;
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE (4096)
void rm_filex_levelx_nor_format_example (void)
{
 spi_flash_instance_t * p_spi_flash_instance = (spi_flash_instance_t *)
g_levelx_nor_spi0_cfg.p_lower_lvl;
 spi_flash_status_t    status;
 /* Erase flash prior to usage */
 fsp_err_t err = p_spi_flash_instance->p_api->open(p_spi_flash_instance->p_ctrl,
p_spi_flash_instance->p_cfg);
    assert(FSP_SUCCESS == err);
 for (uint32_t i = g_levelx_nor_spi0_cfg.address_offset;
         i < g_levelx_nor_spi0_cfg.size;
         i += RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE)
    {
       err = p_spi_flash_instance->p_api->erase(p_spi_flash_instance->p_ctrl,
                                                (uint8_t *)
g_levelx_nor_spi0_cfg.base_address + i,

RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE);
```

```
     assert(FSP_SUCCESS == err);
      status.write_in_progress = true;
 while (status.write_in_progress)
     {
           err =
p_spi_flash_instance->p_api->statusGet(p_spi_flash_instance->p_ctrl, &status);
     assert(FSP_SUCCESS == err);
      }
   }
   err = p_spi_flash_instance->p_api->close(p_spi_flash_instance->p_ctrl);
   assert(FSP_SUCCESS == err);
 /* Format the media */
   UINT fx_err = fx_media_format(&g_fx_media0,
// Pointer to FileX media control block.
 RM_FILEX_LEVELX_NOR_DeviceDriver,                  // Driver entry
                                &g_filex_levelx_nor0, // Pointer to Block Media
Driver
                                g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                  // Volume Name
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS,
// Number of FATs
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS,         // Total sectors
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS,
// Heads
```

```
                                    RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

### Callback Wait Example

This shows how to use the I/O driver callback with ThreadX in order to wait for operations to complete.

```
/* Callback called by FileX block media I/O driver needs to wait on operation. */
void rm_filex_levelx_nor_test_callback_wait (rm_filex_levelx_nor_callback_args_t *
p_args)
{
 if (p_args->event & RM_FILEX_LEVELX_NOR_EVENT_BUSY)
    {
 /* Put the thread to sleep while waiting for operation to complete. */
       tx_thread_sleep(1);
    }
}
void rm_filex_levelx_nor_callback_wait_example (void)
{
 /* Format the media */
    UINT fx_err = fx_media_format(&g_fx_media0,
// Pointer to FileX media control block.
 RM_FILEX_LEVELX_NOR_DeviceDriver,                  // Driver entry
                                &g_filex_levelx_nor0, // Pointer to Block Media
Driver
                                g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                  // Volume Name
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS,
// Number of FATs
                                 RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
```

```
                                    RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS,          // Total sectors
                                    RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER,    // Sectors per cluster
                                    RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS,
// Heads
                                    RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

## Data Structures

| | |
|---|---|
| struct | rm_filex_levelx_nor_callback_args_t |
| struct | rm_filex_levelx_nor_cfg_t |
| struct | rm_filex_levelx_nor_instance_ctrl_t |
| struct | rm_filex_levelx_nor_instance_t |

## Macros

| | |
|---|---|
| #define | VOID |

## Enumerations

| | |
|---|---|
| enum | rm_filex_levelx_nor_event_t |

## Data Structure Documentation

### ◆ rm_filex_levelx_nor_callback_args_t

| struct rm_filex_levelx_nor_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rm_filex_levelx_nor_event_t | event | The event can be used to identify what caused the callback. |

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |

### ◆ rm_filex_levelx_nor_cfg_t

| struct rm_filex_levelx_nor_cfg_t | |
|---|---|
| FileX LevelX configuration | |
| **Data Fields** | |
| UINT(* | nor_driver_initialize )(LX_NOR_FLASH *) |
| | Pointer to the initialization function. |
| | |
| LX_NOR_FLASH * | p_nor_flash |
| | NOR Flash instance. |
| | |
| CHAR * | p_nor_flash_name |
| | NOR Flash instance name. |
| | |
| fsp_err_t(* | close )() |
| | Pointer to underlying driver close. |
| | |
| void(* | p_callback )(rm_filex_levelx_nor_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |

### ◆ rm_filex_levelx_nor_instance_ctrl_t

| struct rm_filex_levelx_nor_instance_ctrl_t | | |
|---|---|---|
| FileX block media private control block. DO NOT MODIFY. Initialization occurs when RM_FILEX_LEVELX_NOR_Open is called. | | |
| Data Fields | | |
| rm_filex_levelx_nor_cfg_t const * | p_cfg | Pointer to instance configuration. |

### ◆ rm_filex_levelx_nor_instance_t

| struct rm_filex_levelx_nor_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_filex_levelx_nor_instance_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_filex_levelx_nor_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |

## Macro Definition Documentation

### ◆ VOID

| #define VOID |
|---|
| Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. |

## Enumeration Type Documentation

### ◆ rm_filex_levelx_nor_event_t

| enum rm_filex_levelx_nor_event_t | |
|---|---|
| Options for the callback events. | |
| Enumerator | |
| RM_FILEX_LEVELX_NOR_EVENT_BUSY | Pending operation, user can define their own wait functionality. |

## Function Documentation

### ◆ RM_FILEX_LEVELX_NOR_DeviceDriver()

void RM_FILEX_LEVELX_NOR_DeviceDriver ( FX_MEDIA * *p_fx_media*)

Access LevelX NOR device functions open, close, read, write and control.

The RM_FILEX_LEVELX_NOR_DeviceDriver function is called from the FileX file system driver and issues requests to a LevelX NOR device through the LevelX API.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_fx_media | FileX media control block. All information about each open media device are maintained in the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide. |

**Return values**

| | |
|---|---|
| None | |

**Returns**

Nothing, but updates FileX media control block.

## 5.2.17.8 Flash (r_flash_hp)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_FLASH_HP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_FLASH_HP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes) |
| fsp_err_t | R_FLASH_HP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks) |
| fsp_err_t | R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *p_blank_check_result) |
| fsp_err_t | R_FLASH_HP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t |

| | |
|---|---|
| | *const p_status) |
| fsp_err_t | R_FLASH_HP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) |
| fsp_err_t | R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| fsp_err_t | R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_Reset (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| fsp_err_t | R_FLASH_HP_BankSwap (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) |
| fsp_err_t | R_FLASH_HP_Close (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_HP_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_FLASH_HP_AntiRollbackCounterIncrement (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_FLASH_HP_AntiRollbackCounterRefresh (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_FLASH_HP_AntiRollbackCounterRead (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter, uint32_t *const p_count) |
| fsp_err_t | R_FLASH_HP_UserLockableAreaWrite (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes) |

### Detailed Description

Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface.

# Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and

ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

### Features

The R_FLASH_HP module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank-checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for ROM Flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.
- Anti-rollback counter support (select MCUs).
- Blocking writing of user lockable data flash option setting memory area (select MCUs).

# Configuration

### Build Time Configurations for r_flash_hp

The following build time configurations are defined in fsp_cfg/r_flash_hp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Code Flash Programming Enable | • Enabled<br>• Disabled | Disabled | Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API. |
| Data Flash Programming Enable | • Enabled<br>• Disabled | Enabled | Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API. |

### Configurations for Storage > Flash (r_flash_hp)

This module can be added to the Stacks tab via New Stack > Storage > Flash (r_flash_hp). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_flash0 | Module name. |
| Data Flash Background | • Enabled | Enabled | Enabling allows Flash |

| | | | |
|---|---|---|---|
| Operation | • Disabled | | API calls that reference data-flash to return immediately, with the operation continuing in the background. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified. Callback function called when a Data Flash Background Operation completes or errors. |
| Flash Ready Interrupt Priority | MCU Specific Options | | Select the flash ready interrupt priority. |
| Flash Error Interrupt Priority | MCU Specific Options | | Select the flash error interrupt priority. |

**Clock Configuration**

Flash uses FCLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

Warning

It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_hp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

**Data Flash Background Operation (BGO) Precautions**

When using the data flash BGO (Background Operation) mode, you can still access the user ROM, RAM and external memory. You must ensure that the data flash is not accessed during a data flash operation. This includes interrupts that may access the data flash.

Data flash BGO is not supported during anti-rollback counter operations or writes to the user lockable area.

**Code Flash Precautions**

Code flash cannot be accessed while writing, erasing or blank checking code flash. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash all supporting code must reside in RAM. This is only done when code flash programming is enabled. BGO mode is not supported for code flash, so a code flash operation will not return before the operation has completed. By default, the vector table resides in the code flash. If an interrupt occurs during the code flash operation, then code flash will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code flash operations. Another option is to copy the vector table to

RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must insure that if in a multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

### Flash Clock (FCLK)

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

### Anti Rollback Counter

Anti-rollback counters are available on select MCUs and can be used to prevent or limit firmware downgrade. The counters can only be incremented and never decremented or reset, even with an external programming tool.

*Note*

> ***The OEM bootloader anti-rollback counter is protected by the security engine and cannot be incremented past the version number specified in the code certificate. This counter works with the First Stage Bootloader (FSBL) set to enabled. For more information, read the 'Secure Boot' section in the 'Security Features' chapter of the user manual of the target MCU and refer to Application Note R11AN0774.***

### Anti-Rollback Counter Configuration Setting

When anti-rollback counters are used for one or more non-secure applications, the Anti-Rollback Counter Configuration Setting for ARC_NSEC (ARCCS) must be configured as either a single line (1xn bits) or multi-line counter (mxn bits). Single line is suitable when there is only one non-secure application using the anti-rollback counter. Multi-line is suitable when there are multiple non-secure applications. Before the counter is configured, it can't be used.

This configuration setting is permanent and must be performed with an external programming tool such as Renesas Flash Programmer (RFP). It cannot be configured in e2 studio.

The secure application and OEM bootloader anti-rollback counters do not require any initial configuration.

### Anti-Rollback Counter Lock Setting

Select MCUs have the ability to permanently lock the anti-rollback counters using the Anti-Rollback Counter Lock Setting (ARCLS). It cannot be set with e2 studio, but if needed, it can be configured with an external programming tool such as Renesas Flash Programmer (RFP). **Once the lock protection is applied to a counter, the counter value can never be incremented again.** It is still possible to read the value of a locked counter.

### Anti-Rollback Counter Recovery from Power Loss or other Counter Increment Error

If an error is returned by R_FLASH_HP_AntiRollbackCounterIncrement or power loss occurs during increment command processing, a successful call to R_FLASH_HP_AntiRollbackCounterRefresh is required before reading or incrementing the counter again. Otherwise, the counter flash may be in an invalid state.

### User Lockable Area

The user lockable area of the data flash configuration area neither requires nor supports erase before rewriting.

Locked sections of the user lockable area cannot be rewritten.

If lockable area is available on the MCU, start address, end address, and write size can be read at runtime in the second element of the data flash block information array (p_info->data_flash.p_block_array[1]) returned by R_FLASH_HP_InfoGet.

### Interrupts

Enable the flash ready interrupt only if you plan to use the data flash BGO. In this mode, the application can initiate a data flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, flash_api_t::FLASH_EVENT_ERASE_COMPLETE) When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the flash_api_t::open API.

*Note*

*The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).*

### Viewing flash contents in e2 studio

By default, the contents of data flash and code flash are cached by e$^2$ studio. This means that during a debug session, modifications to these memory regions will not be observed by e$^2$ studio. When debugging applications using e$^2$ studio, disable the "Allow caching of flash contents" option in the debug configuration in order to view modified flash contents (Debug Configuration > Debugger > Debug Tool Settings > Allow caching of flash contents).

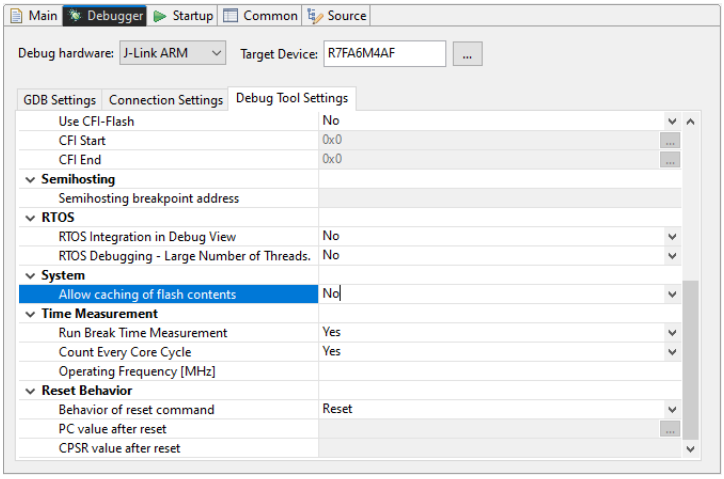



Figure 328: Debug Configuration

### Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is

still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased data flash blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.
- Runtime locking of the user lockable area is not currently supported. The area can be locked by an external programming tool. Do not write to locked areas of user lockable area.

# Examples

### High-Performance Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 64 B: 0x40100000 - 0x4010003F */
#define FLASH_CF_BLOCK_8 0x00010000  /* 32 KB: 0x00010000 - 0x00017FFF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_HP_EXAMPLE_WRITE_SIZE 32
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
void r_flash_hp_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
    assert(FSP_SUCCESS == err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 /* Check if block 0 is erased. */
    err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
 /* Verify the previously erased area is blank */
    assert(FLASH_RESULT_BLANK == blank_check_result);
```

```
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE));
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_8, 1);
    assert(FSP_SUCCESS == err);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_8,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_8,
FLASH_HP_EXAMPLE_WRITE_SIZE));
}
```

### High-Performance Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
```

```
    .irq            = BSP_VECTOR_FLASH_HP_FRDYI_ISR,
};
void r_flash_hp_bgo_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
    interrupt_called = false;
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 while (!interrupt_called)
    {
       ;
    }
    assert(FLASH_EVENT_ERASE_COMPLETE == flash_event);
    interrupt_called = false;
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 flash_status_t status;
 /* Wait until the current flash operation completes. */
 do
    {
       err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
    } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
 /* If the interrupt wasn't called process the error. */
```

```
    assert(interrupt_called);
 /* If the event wasn't a write complete process the error. */
    assert(FLASH_EVENT_WRITE_COMPLETE == flash_event);
 /* Verify the data was written correctly. */
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE));
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

### High-Performance Flash Bank Swap Example

This example demonstrates swapping which flash bank is located at address 0. This feature is only on select MCUs.

```
void r_flash_hp_bankswap_example (void)
{
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Write the new application starting at 0x00200000. */
 /* Swap the block at address 0 with the one at 0x00200000 after the next restart.
  * The application at 0x00200000 must be written there by application code. */
    err = R_FLASH_HP_BankSwap(&g_flash_ctrl);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Handle any pre-reset operations here */
 /* Reset the MCU to swap to the other bank */
    __NVIC_SystemReset();
}
```

### High-Performance Anti-Rollback Counter Basic Example

This example demonstrates basic usage of the anti-rollback counter functions.

```
void r_flash_hp_anti_rollback_counter_example (void)
{
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
    uint32_t current_count;
 /* Read the current counter value for the secure application */
    err = R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_SEC,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the OEM bootloader */
    err = R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_OEMBL,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the non-secure application if counter is
configured as 1xn bits */
    err = R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for 4 non-secure applications if counter is
configured as mxn bits */
    err = R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App 0
    err |= R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_NSEC_1,
&current_count); // NSEC App 1
    err |= R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_NSEC_2,
&current_count); // NSEC App 2
    err |= R_FLASH_HP_AntiRollbackCounterRead(&g_flash_ctrl, FLASH_ARC_NSEC_3,
&current_count); // NSEC App 3
    assert(FSP_SUCCESS == err);
 /* Increment the counter value for the secure application. Caution: The increment
```

```
cannot be reversed. */
    err = R_FLASH_HP_AntiRollbackCounterIncrement(&g_flash_ctrl, FLASH_ARC_SEC);
 /* If error occurred during increment, refresh the counter flash to ensure a valid
state */
 if (FSP_SUCCESS != err)
    {
       err = R_FLASH_HP_AntiRollbackCounterRefresh(&g_flash_ctrl, FLASH_ARC_SEC);
      assert(FSP_SUCCESS == err);
    }
    assert(FSP_SUCCESS == err);
}
```

### High-Performance User Lockable Area Basic Example

This example demonstrates writing the user lockable area.

```
void r_flash_hp_user_lockable_area_example (void)
{
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
    assert(FSP_SUCCESS == err);
 flash_info_t flash_info;
    err = R_FLASH_HP_InfoGet(&g_flash_ctrl, &flash_info);
    assert(FSP_SUCCESS == err);
    assert(flash_info.data_flash.num_regions >= 2);
 flash_block_info_t const * const p_user_area_info =
&flash_info.data_flash.p_block_array[1];
 /* Select initial value to write to the lockable area */
    uint32_t update_complete_flag = 0;
 /* Set parameters for writing update_complete_flag to start of lockable area */
    uint32_t const flash_address    = p_user_area_info->block_section_st_addr;
    uint32_t const write_size_bytes = sizeof(update_complete_flag);
 /* Ensure write size is a multiple of the programming size */
    uint32_t const programming_size_bytes = p_user_area_info->block_size_write;
    assert(write_size_bytes % programming_size_bytes == 0);
```

```
 /* Write data to the beginning of User Lockable Area 0 */
    err = R_FLASH_HP_UserLockableAreaWrite(&g_flash_ctrl,
                                           (uint32_t) &update_complete_flag,
                                           flash_address,
                                           write_size_bytes);
    assert(FSP_SUCCESS == err);
 /* Confirm the data was written */
    assert(0 == memcmp(&update_complete_flag, (uint8_t *) flash_address,
write_size_bytes));
 /* Select new value to write to the lockable area */
    update_complete_flag = 1;
 /* Write new data to the beginning of User Lockable Area 0 */
    err = R_FLASH_HP_UserLockableAreaWrite(&g_flash_ctrl,
                                           (uint32_t) &update_complete_flag,
                                           flash_address,
                                           write_size_bytes);
    assert(FSP_SUCCESS == err);
 /* Confirm the data was written */
    assert(0 == memcmp(&update_complete_flag, (uint8_t *) flash_address,
write_size_bytes));
}
```

### Data Structures

| | |
|---|---|
| struct | flash_hp_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | flash_bgo_operation_t |

## Data Structure Documentation

### ◆ flash_hp_instance_ctrl_t

| struct flash_hp_instance_ctrl_t | |
|---|---|
| Flash HP instance control block. DO NOT INITIALIZE. | |
| **Data Fields** | |
| uint32_t | opened |
| | To check whether api has been opened or not. |

| | |
|---|---|
| flash_bgo_operation_t | current_operation |
| | Operation in progress, for example, FLASH_OPERATION_CF_ERASE. |

### Enumeration Type Documentation

#### ◆ flash_bgo_operation_t

| enum flash_bgo_operation_t |
|---|
| Possible Flash operation states |

### Function Documentation

#### ◆ R_FLASH_HP_Open()

fsp_err_t R_FLASH_HP_Open ( flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg* )

Initializes the high performance flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

Example:

```
/* Open the flash hp instance. */
fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ALREADY_OPEN | The flash control block is already open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |
| FSP_ERR_IRQ_BSP_DISABLED | Caller is requesting BGO but the Flash interrupts are not enabled. |
| FSP_ERR_FCLK | FCLK must be a minimum of 4 MHz for Flash operations. |

### ◆ R_FLASH_HP_Write()

fsp_err_t R_FLASH_HP_Write ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes* )

Writes to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. If BGO is enabled this means the operation was started successfully. |
| FSP_ERR_IN_USE | The Flash peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was not a multiple of the programming size or exceeded the maximum range. |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_FLASH_HP_Erase()

fsp_err_t R_FLASH_HP_Erase ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks* )

Erases the specified Code or Data Flash blocks. Implements flash_api_t::erase by the block_erase_address.

***Note***

*Code flash may contain blocks of different sizes. When erasing code flash it is important to take this into consideration to prevent erasing a larger address space than desired.*

Example:

```
/* Erase 1 block of data flash starting at block 0. */
   err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks specified |
| FSP_ERR_INVALID_ADDRESS | Invalid address specified. If the address is in code flash then code flash programming must be enabled. |
| FSP_ERR_IN_USE | Other flash operation in progress, or API not initialized |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_ERASE_FAILED | Status is indicating a Erase error. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_FLASH_HP_BlankCheck()

fsp_err_t R_FLASH_HP_BlankCheck ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result* )

Performs a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
   err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Blank check operation completed with result in p_blank_check_result, or blank check started and in-progess (BGO mode). |
| FSP_ERR_INVALID_ADDRESS | Invalid data flash address was input. |
| FSP_ERR_INVALID_SIZE | 'num_bytes' was either too large or not aligned for the CF/DF boundary size. |
| FSP_ERR_IN_USE | Other flash operation in progress or API not initialized. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |
| FSP_ERR_BLANK_CHECK_FAILED | Blank check operation failed. |

### ◆ R_FLASH_HP_StatusGet()

fsp_err_t R_FLASH_HP_StatusGet ( flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status* )

Query the FLASH peripheral for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | FLASH peripheral is ready to use. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |

### ◆ R_FLASH_HP_IdCodeSet()

fsp_err_t R_FLASH_HP_IdCodeSet ( flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode* )

Implements flash_api_t::idCodeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID Code successfully configured. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_AccessWindowSet()

fsp_err_t R_FLASH_HP_AccessWindowSet ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr* )

Configure an access window for the Code Flash memory using the provided start and end address. An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block –> last block inclusive. Anything outside this range of Code Flash is then write protected.

***Note***

*If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_HP_AccessWindowClear().*

Implements flash_api_t::accessWindowSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully configured. |
| FSP_ERR_INVALID_ADDRESS | Invalid settings for start_addr and/or end_addr. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_AccessWindowClear()

fsp_err_t R_FLASH_HP_AccessWindowClear ( flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements flash_api_t::accessWindowClear.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully removed. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_Reset()

fsp_err_t R_FLASH_HP_Reset ( flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash circuit successfully reset. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_StartUpAreaSelect()

fsp_err_t R_FLASH_HP_StartUpAreaSelect ( flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary* )

Selects which block, Default (Block 0) or Alternate (Block 1), is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window. Implements flash_api_t::startupAreaSelect.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_BankSwap()

fsp_err_t R_FLASH_HP_BankSwap ( flash_ctrl_t *const *p_api_ctrl*)

Swaps the flash bank located at address 0x00000000 and address 0x00200000. This can only be done when in dual bank mode. Dual bank mode can be enabled in the FSP Configuration Tool under BSP Properties. After a bank swap is done the MCU will need to be reset for the changes to take place. flash_api_t::bankSwap.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Code Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_INVALID_MODE | Cannot switch banks while flash is in Linear mode. |
| FSP_ERR_WRITE_FAILED | Flash write operation failed. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command. |

### ◆ R_FLASH_HP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_HP_UpdateFlashClockFreq ( flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_FCLK | FCLK is not within the acceptable range. |

### ◆ R_FLASH_HP_InfoGet()

fsp_err_t R_FLASH_HP_InfoGet ( flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info* )

Returns the information about the flash regions. Implements flash_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful retrieved the request information. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_info. |

### ◆ R_FLASH_HP_Close()

fsp_err_t R_FLASH_HP_Close ( flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent Flash operations. Implements flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |

### ◆ R_FLASH_HP_CallbackSet()

fsp_err_t R_FLASH_HP_CallbackSet ( flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void *const *p_context*, flash_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements flash_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_FLASH_HP_AntiRollbackCounterIncrement()

fsp_err_t R_FLASH_HP_AntiRollbackCounterIncrement ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Increments the selected anti-rollback counter.

**Warning**

If this function returns an error code other than FSP_SUCCESS, or power loss occurs during a call to this function, call R_FLASH_HP_AntiRollbackCounterRefresh to ensure the counter flash is in a valid state following the error.

***Note***

*The counter is read internally before increment.*

Implements flash_api_t::antiRollbackCounterIncrement

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter incremented successfully |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_api_ctrl |
| FSP_ERR_NOT_ENABLED | The specified counter has not been configured (configuration is only required for ARC_NSEC) |
| FSP_ERR_OVERFLOW | The counter cannot be incremented because it is already at its max value |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Data Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command |
| FSP_ERR_UNSUPPORTED | Data flash programming is not enabled or selected anti-rollback counter is not supported on this MCU |

### ◆ R_FLASH_HP_AntiRollbackCounterRefresh()

fsp_err_t R_FLASH_HP_AntiRollbackCounterRefresh ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Refreshes the selected anti-rollback counter flash area to ensure the flash is in a valid state even if an error occurred during counter increment processing.

This function must be called if errors or power failure occurred during counter increment.

Power failure can be detected by the application code by management of user-defined non-voltile flags during counter increment.

Until a refresh completes successfully, the value of the anti-rollback counter cannot be guaranteed and should not be read.

Implements flash_api_t::antiRollbackCounterRefresh

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter refreshed successfully |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_ENABLED | The specified counter has not been configured (configuration is only required for ARC_NSEC) |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Data Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command |
| FSP_ERR_UNSUPPORTED | Data flash programming is not enabled or Anti-Rollback counter is not supported on this MCU |

### ◆ R_FLASH_HP_AntiRollbackCounterRead()

fsp_err_t R_FLASH_HP_AntiRollbackCounterRead ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*, uint32_t *const *p_count* )

Reads the selected anti-rollback counter and returns the number of counter bits set.

Implements flash_api_t::antiRollbackCounterRead

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter read successfully into p_count |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_api_ctrl or p_count |
| FSP_ERR_NOT_ENABLED | The specified counter has not been configured (configuration is only required for ARC_NSEC) |
| FSP_ERR_PE_FAILURE | Failed to enter or exit Data Flash P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of having received an illegal command |
| FSP_ERR_UNSUPPORTED | Data flash programming is not enabled or Anti-Rollback counter is not supported on this MCU |

### ◆ R_FLASH_HP_UserLockableAreaWrite()

fsp_err_t R_FLASH_HP_UserLockableAreaWrite ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes* )

Write to the user lockable area in flash

Implements flash_api_t::userLockableAreaWrite

***Note***

*BGO is not supported for user lockable area flash operations.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. |
| FSP_ERR_IN_USE | The Flash peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_CMD_LOCKED | FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was not a multiple of the programming size or exceeded the maximum range. |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

## 5.2.17.9 Flash (r_flash_lp)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_FLASH_LP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_FLASH_LP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes) |
| fsp_err_t | R_FLASH_LP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const |

| | |
|---|---|
| | address, uint32_t const num_blocks) |
| fsp_err_t | R_FLASH_LP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *p_blank_check_result) |
| fsp_err_t | R_FLASH_LP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status) |
| fsp_err_t | R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| fsp_err_t | R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) |
| fsp_err_t | R_FLASH_LP_Reset (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| fsp_err_t | R_FLASH_LP_BankSwap (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) |
| fsp_err_t | R_FLASH_LP_Close (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_FLASH_LP_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_FLASH_LP_AntiRollbackCounterIncrement (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_FLASH_LP_AntiRollbackCounterRefresh (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_FLASH_LP_AntiRollbackCounterRead (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter, uint32_t *const p_count) |
| fsp_err_t | R_FLASH_LP_UserLockableAreaWrite (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes) |

## Detailed Description

Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface.

# Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and code flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

### Features

The Low-Power Flash HAL module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for code flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

# Configuration

### Build Time Configurations for r_flash_lp

The following build time configurations are defined in fsp_cfg/r_flash_lp_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Code Flash Programming | • Enabled<br>• Disabled | Disabled | Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API. |
| Code Flash Background Operation Support With Bank Programming | MCU Specific Options | | Enables execution from the startup code flash bank while programming the other code flash bank. |
| Data Flash Programming | • Enabled<br>• Disabled | Enabled | Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API. |
| Data Flash Background Operation Support | • Enabled<br>• Disabled | Enabled | Controls whether or not Data Flash Background |

| | | | |
|---|---|---|---|
| | | | Operation support is included in the build. Disabling reduces the amount of ROM used by the API. |
| Instant Bank Swap (No Reset Required) | MCU Specific Options | | Calls to R_FLASH_LP_BankSwap will immediately swap banks without requiring a reset. |

**Configurations for Storage > Flash (r_flash_lp)**

This module can be added to the Stacks tab via New Stack > Storage > Flash (r_flash_lp).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_flash0 | Module name. |
| Data Flash Background Operation | • Enabled<br>• Disabled | Enabled | Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified. Callback function called when a Data Flash Background Operation completes or errors. |
| Flash Ready Interrupt Priority | MCU Specific Options | | Select the flash ready interrupt priority. |

**Clock Configuration**

Flash either uses FCLK or ICLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

**Pin Configuration**

This module does not use I/O pins.

# Usage Notes

Warning

It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_lp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

### Data Flash Background Operation (BGO) Precautions

When using the data flash BGO, the code flash, RAM and external memory can still be accessed. You must ensure that the data flash is not accessed during a data flash operation. This includes interrupts that may access the data flash.

### Code Flash Precautions

Modifying code flash has limitations requiring some of the flash code to reside in RAM. The use of Bank Programming can limit these requirements but some still exist.

Code flash cannot be accessed while writing, erasing or blank checking code flash unless Bank Programming is supported. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash or flash settings all supporting code must reside in RAM. This is only done when code flash programming is enabled.

When Bank Programming is disabled or not supported a code flash write, erase or blank check operation will not return before the operation has completed.

By default, the vector table resides in the code flash. If an interrupt occurs during the code flash operation other thank a Bank Programming operation, then code flash will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code flash operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must ensure that if in a multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

### Bank Programming Precautions

Bank programming is supported on select MCUs. When enabled Bank1 can be programmed while executing out of Bank0, decreasing the RAM used by the flash driver. When this mode is enabled only Bank1 can be modified.

### Flash Clock Source

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

### Interrupts

Enable the flash ready interrupt only if you plan to use the data flash BGO. In this mode, the application can initiate a data flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, flash_api_t::FLASH_EVENT_ERASE_COMPLETE) When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the flash_api_t::open API.

*Note*

*The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).*

### Viewing flash contents in e2 studio

By default, the contents of data flash and code flash are cached by e$^2$ studio. This means that during a debug session, modifications to these memory regions will not be observed by e$^2$ studio. When debugging applications using e$^2$ studio, disable the "Allow caching of flash contents" option in the debug configuration in order to view modified flash contents (Debug Configuration > Debugger > Debug Tool Settings > Allow caching of flash contents).

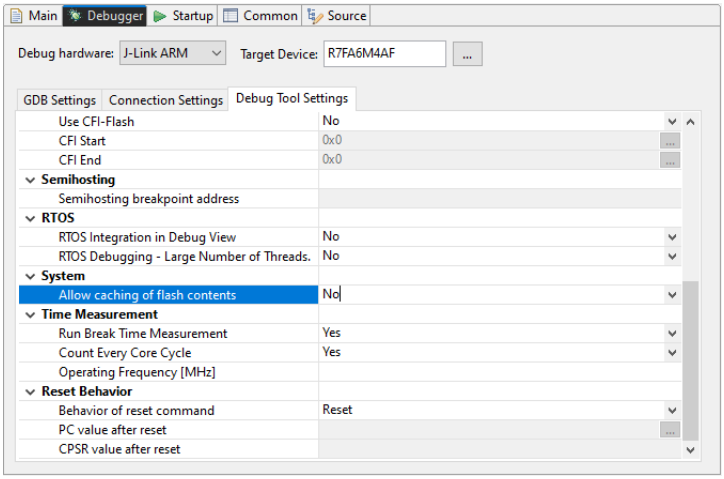



Figure 329: Debug Configuration

### Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.

## Examples

### Low-Power Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 1 KB: 0x40100000 - 0x401003FF */
#define FLASH_CF_BLOCK_10 0x00005000  /* 2 KB: 0x00005000 - 0x000057FF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_LP_EXAMPLE_WRITE_SIZE 32
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
```

```
void R_FLASH_LP_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
    assert(FSP_SUCCESS == err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 /* Check if block 0 is erased. */
    err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
 /* Verify the previously erased area is blank */
    assert(FLASH_RESULT_BLANK == blank_check_result);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE));
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_10, 1);
    assert(FSP_SUCCESS == err);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_10,
FLASH_LP_EXAMPLE_WRITE_SIZE);
```

```
    assert(FSP_SUCCESS == err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_10,
FLASH_LP_EXAMPLE_WRITE_SIZE));
}
```

### Low-Power Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
    .irq            = BSP_VECTOR_FLASH_LP_FRDYI_ISR,
};
void R_FLASH_LP_bgo_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    interrupt_called = false;
 /* Erase 1 block of data flash starting at block 0. */
```

```
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 while (!interrupt_called)
    {
       ;
    }
    assert(FLASH_EVENT_ERASE_COMPLETE == flash_event);
    interrupt_called = false;
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 flash_status_t status;
 /* Wait until the current flash operation completes. */
 do
    {
       err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);
    } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
 /* If the interrupt wasn't called process the error. */
    assert(interrupt_called);
 /* If the event wasn't a write complete process the error. */
    assert(FLASH_EVENT_WRITE_COMPLETE == flash_event);
 /* Verify the data was written correctly. */
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE));
 /* When Bank Programming is enabled the code flash P/E operations are also non
blocking. */
    interrupt_called = false;
 /* Erase the first block of the flash bank not configured as the startup bank. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, (BSP_FEATURE_FLASH_CODE_FLASH_START +
BSP_ROM_SIZE_BYTES) / 2, 1);
    assert(FSP_SUCCESS == err);
 while (!interrupt_called)
    {
```

```
        ;
    }
    assert(FLASH_EVENT_ERASE_COMPLETE == flash_event);
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

### Low-Power Flash Bank Swap Example

This example demonstrates swapping which flash bank is located at address 0. This feature is only on select MCUs.

```
void R_FLASH_LP_bankswap_example (void)
{
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Write the new application starting at 0x0004_0000. */
 /* Swap the bank at address 0 with the one at 0x0004_0000 after the next restart.
  * The application at 0x0004_0000 must be written there by application code. */
    err = R_FLASH_LP_BankSwap(&g_flash_ctrl);
 /* When using Instant Bank Swap the MCU will now have the banks swapped. */
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Handle any pre-reset operations here */
 /* When using Bank Swap with Reset the MCU must be reset to swap to the other bank
*/
    __NVIC_SystemReset();
}
```

## Data Structures

| | |
|---|---|
| struct | flash_lp_instance_ctrl_t |

## Data Structure Documentation

### ◆ flash_lp_instance_ctrl_t

| struct flash_lp_instance_ctrl_t |
| --- |
| Flash instance control block. DO NOT INITIALIZE. Initialization occurs when R_FLASH_LP_Open() is called. |

## Function Documentation

### ◆ R_FLASH_LP_Open()

fsp_err_t R_FLASH_LP_Open ( flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg* )

Initialize the Low Power flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

This function must be called once prior to calling any other FLASH API functions. If a user supplied callback function is supplied, then the Flash Ready interrupt will be configured to call the users callback routine with an Event type describing the source of the interrupt for Data Flash operations.

Example:

```
/* Open the flash lp instance. */
fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
```

***Note***

*Providing a callback function in the supplied p_cfg->callback field automatically configures the Flash for Data Flash to operate in non-blocking background operation (BGO) mode.*

**Return values**

| | |
| --- | --- |
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl, p_cfg or p_callback if BGO is enabled. |
| FSP_ERR_IRQ_BSP_DISABLED | Caller is requesting BGO but the Flash interrupts are not enabled. |
| FSP_ERR_FCLK | FCLK must be a minimum of 4 MHz for Flash operations. |
| FSP_ERR_ALREADY_OPEN | Flash Open() has already been called. |
| FSP_ERR_TIMEOUT | Failed to exit P/E mode after configuring flash. |
| FSP_ERR_INVALID_STATE | The system is not running from the required clock or the required clock is not running and stable. |

### ◆ R_FLASH_LP_Write()

fsp_err_t R_FLASH_LP_Write ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes* )

Write to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. If BGO is enabled this means the operation was started successfully. |
| FSP_ERR_IN_USE | The Flash peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was not a multiple of the programming size or exceeded the maximum range. |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |

### ◆ R_FLASH_LP_Erase()

fsp_err_t R_FLASH_LP_Erase ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks* )

Erase the specified Code or Data Flash blocks. Implements flash_api_t::erase.

Example:

```
/* Erase 1 block of data flash starting at block 0. */
   err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks specified |
| FSP_ERR_INVALID_ADDRESS | Invalid address specified |
| FSP_ERR_IN_USE | Other flash operation in progress, or API not initialized |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | The Flash API is not Open. |
| FSP_ERR_TIMEOUT | Timed out waiting for FCU to be ready. |
| FSP_ERR_ERASE_FAILED | Status is indicating a Erase error. |

### ◆ R_FLASH_LP_BlankCheck()

fsp_err_t R_FLASH_LP_BlankCheck ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result* )

Perform a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
   err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode). |
| FSP_ERR_INVALID_ADDRESS | Invalid data flash address was input |
| FSP_ERR_INVALID_SIZE | 'num_bytes' was either too large or not aligned for the CF/DF boundary size. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_BLANK_CHECK_FAILED | An error occurred during blank checking. |

### ◆ R_FLASH_LP_StatusGet()

fsp_err_t R_FLASH_LP_StatusGet ( flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status* )

Query the FLASH for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash is ready and available to accept commands. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |

### ◆ R_FLASH_LP_AccessWindowSet()

fsp_err_t R_FLASH_LP_AccessWindowSet ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr* )

Configure an access window for the Code Flash memory. Implements flash_api_t::accessWindowSet .

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block (inclusive) –> last block (exclusive). Anything outside this range of Code Flash is then write protected. As an example, if you wanted to place an accesswindow on Code Flash Blocks 0 and 1, such that only those two blocks were writable, you would need to specify (address in block 0, address in block 2) as the respective start and end address.

***Note***

*If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_LP_AccessWindowClear().*

The invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

**Parameters**

| | | |
|---|---|---|
| | p_api_ctrl | The p api control |
| [in] | start_addr | The start address |
| [in] | end_addr | The end address |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully configured. |
| FSP_ERR_INVALID_ADDRESS | Invalid settings for start_addr and/or end_addr. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_AccessWindowClear()

fsp_err_t R_FLASH_LP_AccessWindowClear ( flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is configured in the Code Flash. Implements flash_api_t::accessWindowClear. On successful return from this call all Code Flash is writable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Access window successfully removed. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_IdCodeSet()

fsp_err_t R_FLASH_LP_IdCodeSet ( flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode* )

Write the ID code provided to the id code registers. Implements flash_api_t::idCodeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ID code successfully configured. |
| FSP_ERR_IN_USE | FLASH peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for completion of extra command. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |

### ◆ R_FLASH_LP_Reset()

fsp_err_t R_FLASH_LP_Reset ( flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Flash circuit successfully reset. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |

### ◆ R_FLASH_LP_StartUpAreaSelect()

fsp_err_t R_FLASH_LP_StartUpAreaSelect ( flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary* )

Select which block is used as the startup area block. Implements flash_api_t::startupAreaSelect.

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |
| FSP_ERR_UNSUPPORTED | Code Flash Programming is not enabled. Cannot set FLASH_STARTUP_AREA_BTFLG when the temporary flag is false. |

### ◆ R_FLASH_LP_BankSwap()

fsp_err_t R_FLASH_LP_BankSwap ( flash_ctrl_t *const *p_api_ctrl*)

Swap the Code Flash bank to update new program. Implement flash_api_t::bankSwap.

Swap the flash bank located at address 0x00000000 and address 0x00040000. After a bank swap is done the MCU will need to be reset for the changes to take place unless instant swap is enabled.

To use this API, Code Flash Programming in the FSP Configuration Tool under Stack Properties must be enabled.

***Note***

*This function only available on MCUs which support bank swap feature.*
*When active bank is bank 1, startup program protection function is invalid.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | The api control instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Banks were swapped. |
| FSP_ERR_UNSUPPORTED | Module does not support Bank Swap. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_IN_USE | Extra area is being used by other command. |

### ◆ R_FLASH_LP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_LP_UpdateFlashClockFreq ( flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | Flash is busy with an on-going operation. |
| FSP_ERR_FCLK | Invalid flash clock source frequency. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_TIMEOUT | Timed out waiting for the FCU to become ready. |

### ◆ R_FLASH_LP_InfoGet()

fsp_err_t R_FLASH_LP_InfoGet ( flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info* )

Returns the information about the flash regions. Implements flash_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful retrieved the request information. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_info. |
| FSP_ERR_NOT_OPEN | The flash is not open. |

### ◆ R_FLASH_LP_Close()

fsp_err_t R_FLASH_LP_Close ( flash_ctrl_t *const *p_api_ctrl*)

Release any resources that were allocated by the Flash API. Implements flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |
| FSP_ERR_NOT_OPEN | Flash API has not yet been opened. |
| FSP_ERR_IN_USE | The flash is currently in P/E mode. |

### ◆ R_FLASH_LP_CallbackSet()

fsp_err_t R_FLASH_LP_CallbackSet ( flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void *const *p_context*, flash_callback_args_t *const *p_callback_memory* )

Stub function Implements flash_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function has not been implemented. |

### ◆ R_FLASH_LP_AntiRollbackCounterIncrement()

fsp_err_t R_FLASH_LP_AntiRollbackCounterIncrement ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Stub function Implements flash_api_t::antiRollbackCounterIncrement.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function has not been implemented. |

### ◆ R_FLASH_LP_AntiRollbackCounterRefresh()

fsp_err_t R_FLASH_LP_AntiRollbackCounterRefresh ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Stub function Implements flash_api_t::antiRollbackCounterRefresh.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function has not been implemented. |

### ◆ R_FLASH_LP_AntiRollbackCounterRead()

fsp_err_t R_FLASH_LP_AntiRollbackCounterRead ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*, uint32_t *const *p_count* )

Stub function Implements flash_api_t::antiRollbackCounterRead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function has not been implemented. |

### ◆ R_FLASH_LP_UserLockableAreaWrite()

fsp_err_t R_FLASH_LP_UserLockableAreaWrite ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes* )

Stub function Implements flash_api_t::userLockableAreaWrite.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function has not been implemented. |

### 5.2.17.10 FreeRTOS+FAT Port for RA (rm_freertos_plus_fat)

Modules » Storage

#### Functions

| | |
|---|---|
| fsp_err_t | RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info) |
| fsp_err_t | RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const p_ctrl) |

#### Detailed Description

Middleware for the FAT File System control on RA MCUs.

# Overview

This module provides the hardware port layer for FreeRTOS+FAT file system. After initializing this module, refer to the FreeRTOS+FAT API reference to use the file system: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_FAT/index.html

### Features

The FreeRTOS+FAT port module supports the following features:

- Callbacks for insertion and removal for removable devices.
- Helper function to initialize FF_Disk_t
- Blocking read and write port functions that use FreeRTOS task notification to pend if FreeRTOS is used
- FreeRTOS is optional

# Configuration

### Build Time Configurations for rm_freertos_plus_fat

The following build time configurations are defined in fsp_cfg/middleware/rm_freertos_plus_fat_cfg.h:

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Storage > FreeRTOS+FAT Port for RA (rm_freertos_plus_fat)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_freertos_plus_fat0 | Module name. |
| Total Number of Sectors | Must be a non-negative integer | 31293440 | Enter the total number of sectors on the device. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit(). |
| Sector Size (bytes) | Must be a power of 2 multiple of 512 | 512 | Select the sector size. Must match the underlying media sector size and at least 512. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit(). |
| Cache Size (bytes) | Must be a power of 2 multiple of 512 | 1024 | Select the cache size. Must be a multiple of the sector size and at least 2 times the sector size. |
| Partition Number | Must be a non-negative integer | 0 | Select the partition number for this disk. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed. |

# Usage Notes

### Pending during Read/Write

If the underlying driver supports non-blocking operations, the FreeRTOS+FAT port pends the active

FreeRTOS task during read and write operations so other tasks can run in the background.

If FreeRTOS is not used, the FreeRTOS+FAT port spins in a while loop waiting for read and write operations to complete.

### FreeRTOS+FAT without FreeRTOS

To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.

Also, update the Malloc function to malloc and the Free function to free in the Common configurations.

# Examples

### Basic Example

This is a basic example of FreeRTOS+FAT in an application.

```
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
extern rm_freertos_plus_fat_instance_ctrl_t g_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_freertos_plus_fat0_cfg;
extern rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat_disk_cfg;
extern uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_freertos_plus_fat_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
```

```
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

### Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_freertos_plus_fat_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Try to mount the disk. If the disk is not formatted, mount will fail. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 if (FF_isERR((uint32_t) ff_err))
    {
 /* The disk is likely not formatted. Partition and format the disk, then mount
again. */
       FF_PartitionParameters_t partition_params;
       partition_params.ulSectorCount   =
g_rm_freertos_plus_fat_disk_cfg.device.sector_count;
       partition_params.ulHiddenSectors = 1;
       partition_params.ulInterSpace    = 0;
       memset(partition_params.xSizes, 0, sizeof(partition_params.xSizes));
       partition_params.xSizes[RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER] =
```

```
            (BaseType_t) partition_params.ulSectorCount - 1;
       partition_params.xPrimaryCount = 1;
       partition_params.eSizeType    = eSizeIsSectors;
       ff_err = FF_Partition(&disk, &partition_params);
      handle_ff_error(ff_err);
       ff_err = FF_Format(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER,
pdFALSE, pdFALSE);
      handle_ff_error(ff_err);
       ff_err = FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
      handle_ff_error(ff_err);
    }
}
```

### Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
#if 2 == BSP_CFG_RTOS
static EventGroupHandle_t xUSBEventGroupHandle = NULL;
#else
volatile uint32_t g_rm_freertos_plus_fat_insertion_events = 0;
volatile uint32_t g_rm_freertos_plus_fat_removal_events = 0;
#endif
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_freertos_plus_fat_test_callback (rm_freertos_plus_fat_callback_args_t *
p_args)
{
#if 2 == BSP_CFG_RTOS
 /* Post an event if FreeRTOS is available. */
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xEventGroupSetBitsFromISR(xUSBEventGroupHandle, p_args->event,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
#else
 /* If FreeRTOS is not used, set a global flag. */
```

```
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED)
    {
       g_rm_freertos_plus_fat_insertion_events++;
    }
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED)
    {
       g_rm_freertos_plus_fat_removal_events++;
    }
#endif
}
void rm_freertos_plus_fat_media_insertion_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
    TEST_ASSERT_NOT_EQUAL(NULL, xUSBEventGroupHandle);
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for media insertion. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
                                                       pdTRUE,
                                                       pdFALSE,
                                                       portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
```

```
 /* Wait for media insertion. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
}
```

### Media Insertion Example for USB

This shows how to use the callback to read and write to USB media.

```
void rm_freertos_plus_fat_usb_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for the USB media to be attached. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
```

```
                                                    pdTRUE,
                                                    pdFALSE,
                                                    portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
 /* Wait for the USB media to be attached. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
```

```
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

## Data Structures

| | |
|---|---|
| struct | rm_freertos_plus_fat_instance_ctrl_t |

## Data Structure Documentation

### ◆ rm_freertos_plus_fat_instance_ctrl_t

| struct rm_freertos_plus_fat_instance_ctrl_t |
|---|
| FreeRTOS plus FAT private control block. DO NOT MODIFY. Initialization occurs when RM_FREERTOS_PLUS_FAT_Open is called. |

## Function Documentation

### ◆ RM_FREERTOS_PLUS_FAT_Open()

fsp_err_t RM_FREERTOS_PLUS_FAT_Open ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_cfg_t const *const *p_cfg* )

Initializes lower layer media device.

Implements rm_freertos_plus_fat_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_OUT_OF_MEMORY | Not enough memory to create semaphore. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- rm_block_media_api_t::open

### ◆ RM_FREERTOS_PLUS_FAT_MediaInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_MediaInit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_device_t *const *p_device* )

Initializes the media device. This function blocks until all identification and configuration commands are complete.

Implements rm_freertos_plus_fat_api_t::mediaInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- rm_block_media_api_t::mediaInit
- rm_block_media_api_t::infoGet

### ◆ RM_FREERTOS_PLUS_FAT_DiskInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskInit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_disk_cfg_t const *const *p_disk_cfg*, FF_Disk_t *const *p_disk* )

Initializes a FreeRTOS+FAT disk structure. This function calls FF_CreateIOManger.

Implements rm_freertos_plus_fat_api_t::diskInit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |
| FSP_ERR_INTERNAL | Call to FF_CreateIOManger failed. |

### ◆ RM_FREERTOS_PLUS_FAT_DiskDeinit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskDeinit ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk* )

Deinitializes a FreeRTOS+FAT disk structure. This function calls FF_DeleteIOManger.

Implements rm_freertos_plus_fat_api_t::diskDeinit().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is initialized and ready to access the memory device. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Module has not been initialized. |

### ◆ RM_FREERTOS_PLUS_FAT_InfoGet()

fsp_err_t RM_FREERTOS_PLUS_FAT_InfoGet ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*, rm_freertos_plus_fat_info_t *const *p_info* )

Get partition information. This function can only be called after rm_freertos_plus_fat_api_t::diskInit() .

Implements rm_freertos_plus_fat_api_t::infoGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information stored in p_info. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |
| FSP_ERR_NOT_FOUND | The value of p_iomanager is NULL. |

### ◆ RM_FREERTOS_PLUS_FAT_Close()

fsp_err_t RM_FREERTOS_PLUS_FAT_Close ( rm_freertos_plus_fat_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_freertos_plus_fat_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- rm_block_media_api_t::close

## 5.2.17.11 LevelX NOR Port (rm_levelx_nor_spi)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_LEVELX_NOR_SPI_Open (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, rm_levelx_nor_spi_cfg_t const *const p_cfg)<br><br>Initializes LevelX NOR SPI port read/write and control. More... |

| | |
|---|---|
| fsp_err_t | RM_LEVELX_NOR_SPI_Read (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG *const p_flash_addr, ULONG *const p_dest, ULONG word_count)<br>LevelX NOR driver "read sector" service. More... |
| fsp_err_t | RM_LEVELX_NOR_SPI_Write (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG *const p_flash_addr, ULONG *const p_src, ULONG word_count)<br>LevelX NOR driver "write sector" service. More... |
| fsp_err_t | RM_LEVELX_NOR_SPI_BlockErase (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG block, ULONG erase_count)<br>LevelX NOR driver "block erase" service. More... |
| fsp_err_t | RM_LEVELX_NOR_SPI_BlockErasedVerify (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG block)<br>LevelX NOR driver "block erased verify" service. More... |
| fsp_err_t | RM_LEVELX_NOR_SPI_Close (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl)<br>LevelX NOR driver close service. More... |

### Detailed Description

Middleware for using Azure RTOS LevelX on NOR SPI memory.

# Overview

This module provides the hardware port layer for LevelX on NOR SPI flash memory. Setup for this module is done solely through calling LevelX APIs. Please refer to the LevelX API reference: https://docs.microsoft.com/en-us/azure/rtos/levelx/

# Configuration

### Build Time Configurations for rm_levelx_nor_spi

The following build time configurations are defined in fsp_cfg/middleware/rm_levelx_nor_spi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | Selects if code for parameter checking is to be included in the |

| | | | |
|---|---|---|---|
| | | | build. |
| Page Buffer Size (bytes) | Size should be greater than zero | 256 | When direct read is enabled in LevelX a situation can occur where the driver has to write to SPI memory with the source locaiton also being within the SPI memory address range. In this situation the driver needs a buffer that is at least the same size as a page in order to temporarily store data to write out. |

**Configurations for Storage > LevelX NOR Port (rm_levelx_nor_spi)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_levelx_nor_spi0 | Module name. |
| Memory Start Address Offset (bytes) | Offset should be greater than or equal to zero | 0 | Enter the starting offset to use in the SPI memory. The starting address for LevelX memory will be the SPI memory base address plus this offset. |
| Memory Size (bytes) | Size should be greater than zero | 33554432 | Enter the size that the LevelX Memory should be. This can be smaller than the SPI memory size in order to use a subset of SPI memory. |
| Poll Status Count | Poll Status Count should be greater than or equal to zero | 0xFFFFFFFF | Number of times to poll for operation complete status for blocking memory operations. |

**Build Time Configurations for lx**

The following build time configurations are defined in fsp_cfg/azure/lx/lx_user.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| NOR | | | |
| Direct Read | • Enabled (default)<br>• Disabled | Enabled (default) | When enabled, this option bypasses the NOR flash driver read routine in favor or |

| | | | |
|---|---|---|---|
| | | | reading the NOR memory directly, resulting in a significant performance increase. |
| Free Sector Data Verify | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, this causes the LevelX NOR instance open logic to verify free NOR sectors are all ones. |
| Extended Cache | • Enabled (default)<br>• Disabled | Enabled (default) | Enables the extended NOR cache. |
| Extended Cache Size | Manual Entry | | If not set this value defaults to 8, which represents a maximum of 8 sectors that can be cached in a NOR instance. |
| Sector Mapping Cache Size | Value must be greater than or equal to 8 and a power of 2, or empty | | If not set this value defaults to 16 and defines the logical sector mapping cache size. Large values improve performance, but cost memory. The minimum size is 8 and all values must be a power of 2. |
| NAND | | | |
| Sector Mapping Cache Size | Value must be greater than or equal to 8 and a power of 2, or empty | | If not set this value defaults to 128 and defines the logical sector mapping cache size. Large values improve performance, but cost memory. The minimum size is 8 and all values must be a power of 2. |
| Flash Direct Mapping Cache | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, this creates a direct mapping cache, such that there are no cache misses. It also required that LX_NAND_SECTOR_MAPPING_CACHE_SIZE represents the exact number of total pages in your flash device. |

| | | | |
|---|---|---|---|
| Thread Safe | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, this makes LevelX thread-safe by using a ThreadX mutex object throughout the API. |
| Standalone Mode | • Enabled<br>• Disabled (default) | Disabled (default) | When enabled, allows LevelX to be used in standalone mode (without Azure RTOS). |

# Usage Notes

### Pending during Erase/Write

The LevelX NOR SPI driver is blocking on all SPI operations and will poll device status for operation completion on Writes and Erases. A callback can be provided by the user to wait with an OS-specific thread wait in these instances.

### Closing the driver

When lx_nor_flash_close is called to close the LevelX instance it does not call any services within the LevelX NOR SPI driver to close out the driver instance. The user should call the generated close function (i.e. g_rm_levelx_nor_spi0_close) in order to close out the driver instance.

### Erasing Flash Memory Prior to Usage

The area of the flash memory being used for the LevelX instance should be erased using the lower level flash API prior to usage. Otherwise, LevelX API may fail on lx_nor_flash_open due to any areas in flash memory that have been written/set.

# Examples

### Basic Example

This is a basic example of using the LevelX NOR SPI driver with the LevelX API in an application.

```
#define RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE (512)
#define RM_LEVELX_NOR_SPI_EXAMPLE_BUFFER_FILL_VALUE (0xA5)
#define RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE (4096)
extern rm_levelx_nor_spi_instance_ctrl_t g_levelx_nor_spi0_ctrl;
extern rm_levelx_nor_spi_cfg_t g_levelx_nor_spi0_cfg;
extern LX_NOR_FLASH g_lx_nor_flash0;
void rm_levelx_nor_spi_example (void)
{
    uint8_t               read_buffer[RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE];
    uint8_t               write_buffer[RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE];
```

```
 spi_flash_instance_t * p_spi_flash_instance = (spi_flash_instance_t *)
g_levelx_nor_spi0_cfg.p_lower_lvl;
 spi_flash_status_t    status;
    memset(write_buffer, RM_LEVELX_NOR_SPI_EXAMPLE_BUFFER_FILL_VALUE, sizeof
(write_buffer));
 /* Erase flash prior to usage */
 fsp_err_t err = p_spi_flash_instance->p_api->open(p_spi_flash_instance->p_ctrl,
p_spi_flash_instance->p_cfg);
    assert(FSP_SUCCESS == err);
 for (uint32_t i = g_levelx_nor_spi0_cfg.address_offset;
         i < g_levelx_nor_spi0_cfg.size;
         i += RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE)
    {
       err = p_spi_flash_instance->p_api->erase(p_spi_flash_instance->p_ctrl,
                                                (uint8_t *)
g_levelx_nor_spi0_cfg.base_address + i,

RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE);
      assert(FSP_SUCCESS == err);
       status.write_in_progress = true;
 while (status.write_in_progress)
      {
           err =
p_spi_flash_instance->p_api->statusGet(p_spi_flash_instance->p_ctrl, &status);
      assert(FSP_SUCCESS == err);
       }
    }
    err = p_spi_flash_instance->p_api->close(p_spi_flash_instance->p_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize LevelX */
    lx_nor_flash_initialize();
    UINT lx_err = lx_nor_flash_open(&g_lx_nor_flash0, "LX_NOR_SPI_EXAMPLE",
g_levelx_nor_spi0_initialize);
    handle_lx_error(lx_err);
```

```
 /* Write test value to sector 0 then read back to verify */
    lx_err = lx_nor_flash_sector_write(&g_lx_nor_flash0, 0, write_buffer);
    handle_lx_error(lx_err);
    lx_err = lx_nor_flash_sector_read(&g_lx_nor_flash0, 0, read_buffer);
    handle_lx_error(lx_err);
    assert(0 == memcmp(read_buffer, write_buffer, sizeof(read_buffer)));
}
```

### Callback Wait Example

This shows how to use the LevelX NOR SPI driver callback with ThreadX in order to wait for operations to complete.

```
/* Callback called by LevelX NOR SPI driver needs to wait on operation. */
void rm_levelx_nor_spi_callback_wait_example (rm_levelx_nor_spi_callback_args_t *
p_args)
{
 if (p_args->event & RM_LEVELX_NOR_SPI_EVENT_BUSY)
    {
 /* Put the thread to sleep while waiting for operation to complete. */
       tx_thread_sleep(1);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | rm_levelx_nor_spi_callback_args_t |
| struct | rm_levelx_nor_spi_cfg_t |
| struct | rm_levelx_nor_spi_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rm_levelx_nor_spi_event_t |

## Data Structure Documentation

### ◆ rm_levelx_nor_spi_callback_args_t

| |
|---|
| struct rm_levelx_nor_spi_callback_args_t |
| RM_LEVELX_NOR_SPI callback arguments definitions |

| Data Fields | | |
|---|---|---|
| rm_levelx_nor_spi_event_t | event | LevelX NOR driver callback event. |
| void * | p_context | Placeholder for user data. |

### ◆ rm_levelx_nor_spi_cfg_t

| struct rm_levelx_nor_spi_cfg_t | |
|---|---|
| SF_EL_LX_NOR Config Block Type | |
| **Data Fields** | |
| spi_flash_instance_t const * | p_lower_lvl |
| | Lower level memory pointer. |
| | |
| LX_NOR_FLASH * | p_lx_nor_flash |
| | Pointer to the LevelX nor flash instance. |
| | |
| uint32_t | base_address |
| | Base address of memory mapped region. |
| | |
| uint32_t | address_offset |
| | Offset to use subset of available flash size if desired. |
| | |
| uint32_t | size |
| | Size of the partitioned region. |
| | |
| uint32_t | poll_status_count |
| | Number of times to poll for operation complete status before returning an error. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback. |
| | |

| | |
|---|---|
| void(* | p_callback )(rm_levelx_nor_spi_callback_args_t *p_args) |
| | Callback function. |
| | |

### ◆ rm_levelx_nor_spi_instance_ctrl_t

| struct rm_levelx_nor_spi_instance_ctrl_t | | |
|---|---|---|
| SF_EL_LX_NOR Control Block Type | | |
| Data Fields | | |
| rm_levelx_nor_spi_cfg_t const * | p_cfg | Pointer to instance configuration. |
| uint32_t | start_address | Start address of partition to use within memory mapped region. |
| uint32_t | minimum_erase_size | Minimum erase size of SPI memory. |
| uint8_t | page_buffer[RM_LEVELX_NOR_SPI_CFG_BUFFER_SIZE] | Page buffer for situations when writing to SPI memory from a source within SPI memory. |
| uint32_t | open | Used to determine if module is initialized. |

## Enumeration Type Documentation

### ◆ rm_levelx_nor_spi_event_t

| enum rm_levelx_nor_spi_event_t | |
|---|---|
| Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. Options for the callback events. | |
| Enumerator | |
| RM_LEVELX_NOR_SPI_EVENT_BUSY | Pending operation, user can define their own wait functionality. |

## Function Documentation

### ◆ RM_LEVELX_NOR_SPI_Open()

fsp_err_t RM_LEVELX_NOR_SPI_Open ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, rm_levelx_nor_spi_cfg_t const *const *p_cfg* )

Initializes LevelX NOR SPI port read/write and control.

Calls lower level SPI memory functions.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Control block for the LevelX NOR SPI instance. |
| [in,out] | p_cfg | Configuration for LevelX NOR SPI port. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX NOR driver is successfully opened. |
| FSP_ERR_ASSERTION | p_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Driver is already in OPEN state. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:open

### ◆ RM_LEVELX_NOR_SPI_Read()

fsp_err_t RM_LEVELX_NOR_SPI_Read ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash_addr*, ULONG *const *p_dest*, ULONG *word_count* )

LevelX NOR driver "read sector" service.

This is responsible for reading a specific sector in a specific block of the NOR flash. All error checking and correcting logic is the responsibility of this service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block for the LevelX NOR SPI instance. |
| [in] | p_flash_addr | Specifies the address of a logical sector within a NOR flash block of memory. |
| [in,out] | p_dest | Specifies where to place the sector contents. |
| [in] | word_count | Specifies how many 32-bit words to read. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX NOR flash sector read successful. |
| FSP_ERR_ASSERTION | p_ctrl, p_flash_addr or p_dest is NULL. |
| FSP_ERR_NOT_OPEN | Driver not in OPEN state for reading. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes.

### ◆ RM_LEVELX_NOR_SPI_Write()

fsp_err_t RM_LEVELX_NOR_SPI_Write ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash_addr*, ULONG *const *p_src*, ULONG *word_count* )

LevelX NOR driver "write sector" service.

This is responsible for writing a specific sector into a block of the NOR flash. All error checking is the responsibility of the this service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block for the LevelX NOR SPI instance. |
| [in,out] | p_flash_addr | Specifies the address of a logical sector within a NOR flash block of memory. |
| [in] | p_src | Specifies the source of the write. |
| [in] | word_count | Specifies how many 32-bit words to write. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX NOR flash sector write successful. |
| FSP_ERR_ASSERTION | p_ctrl, p_flash_addr or p_src is NULL. |
| FSP_ERR_NOT_OPEN | Driver not in OPEN state for writing. |
| FSP_ERR_TIMEOUT | Timeout occurred while waiting for operation to complete. |
| FSP_ERR_WRITE_FAILED | Verification of Write operation failed. |
| FSP_ERR_INVALID_ADDRESS | Write address or size falls outside of flash memory range. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:write

### ◆ RM_LEVELX_NOR_SPI_BlockErase()

fsp_err_t RM_LEVELX_NOR_SPI_BlockErase ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *block*, ULONG *erase_count* )

LevelX NOR driver "block erase" service.

This is responsible for erasing the specified block of the NOR flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block for the LevelX NOR SPI instance. |
| [in] | block | Specifies which NOR block to erase. |
| [in] | erase_count | Provided for diagnostic purposes(currently unused). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX NOR flash block erase successful. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver not in OPEN state for erasing. |
| FSP_ERR_TIMEOUT | Timeout occurred while waiting for operation to complete. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:erase

### ◆ RM_LEVELX_NOR_SPI_BlockErasedVerify()

fsp_err_t RM_LEVELX_NOR_SPI_BlockErasedVerify ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *block* )

LevelX NOR driver "block erased verify" service.

This is responsible for verifying the specified block of the NOR flash is erased.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block for the LevelX NOR SPI instance. |
| [in] | block | Specifies which block to verify that it is erased. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX flash block erase verification successful. |
| FSP_ERR_ASSERTION | p_ctrl or lower level driver is NULL. |
| FSP_ERR_NOT_OPEN | Driver not in OPEN state for verifying. |
| FSP_ERR_NOT_ERASED | The block is not erased properly. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes.

### ◆ RM_LEVELX_NOR_SPI_Close()

fsp_err_t RM_LEVELX_NOR_SPI_Close ( rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*)

LevelX NOR driver close service.

This is responsible for closing the driver properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block for the LevelX NOR SPI instance. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | LevelX flash is available and is now open for read, write, and control access. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver not in OPEN state for closing. |

**Returns**

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:close

### 5.2.17.12 LittleFS on Flash (rm_littlefs_flash)

Modules » Storage

**Functions**

| | |
|---|---|
| fsp_err_t | RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
| fsp_err_t | RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl) |

**Detailed Description**

Middleware for the LittleFS File System control on RA MCUs.

# Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system:
https://github.com/ARMmbed/littlefs

# Configuration

### Build Time Configurations for rm_littlefs_flash

The following build time configurations are defined in fsp_cfg/rm_littlefs_flash_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Storage > LittleFS on Flash (rm_littlefs_flash)

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_littlefs0 | Module name. |
| Read Size | Must be a non-negative integer | 1 | Minimum size of a block read. All read operations will be a multiple of this value. |
| Program Size | Must be a non-negative integer | 4 | Minimum size of a block program. All program operations will be a multiple of this value. |

| | | | |
|---|---|---|---|
| Memory Start Address | Manual Entry | BSP_FEATURE_FLASH_DATA_FLASH_START | Start address of the memory allocated for the LittleFS Volume. |
| Block Size (bytes) | Must be a multiple of 64 | 128 | Size of an erasable block. This does not impact RAM consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes. |
| Block Count | Manual Entry | (BSP_DATA_FLASH_SIZE_BYTES/128) | Number of erasable blocks on the device. |
| Block Cycles | Must be an integer | 1024 | Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling. |
| Cache Size | Must be a non-negative integer | 64 | Size of block caches. Each cache buffers a portion of a block in RAM. The LittleFS needs a read cache, a program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size. |
| Lookahead Size | Must be a non-negative multiple of 8 | 16 | Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number |

| | | | |
|---|---|---|---|
| | | | of blocks found during an allocation pass. The lookahead buffer is stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8. |

### Common LittleFS Configuration

### Build Time Configurations for LittleFS

The following build time configurations are defined in arm/littlefs/lfs_util.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Custom lfs_util.h | Manual Entry | | Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Thread Safe | • Enabled<br>• Disabled | Disabled | Enables thread safety in LittleFS. |
| Read Only | • Enabled<br>• Disabled | Disabled | Enables Read Only mode in LittleFS. |
| Use Malloc | • Enabled<br>• Disabled | Enabled | Configures the use of malloc by LittleFS. |
| Use Assert | • Enabled<br>• Disabled | Enabled | Configures the use of assert by LittleFS. |
| Debug Messages | • Enabled<br>• Disabled | Disabled | Configures debug messages. |
| Warning Messages | • Enabled<br>• Disabled | Disabled | Configures warning messages. |
| Error Messages | • Enabled<br>• Disabled | Disabled | Configures error messages. |
| Trace Messages | • Enabled<br>• Disabled | Disabled | Configures trace messages. |
| Intrinsics | • Enabled<br>• Disabled | Enabled | Configures intrinsic functions such as __builtin_clz. |
| Instance Name for STDIO wrapper | Name must be a valid C symbol | g_rm_littlefs0 | The rm_littlefs instance name to use with the STDIO wrapper. |

# Usage Notes

### Memory Address

LittleFS requrires a section of non volatile memory to store data. It is recommended to use Data Flash when available. In other cases, such as MRAM, a section of memory should be reserved in the Memories Tab using an RA Solution project. The address of this memory section must be entered in the rm_littlefs_flash configuration. If the memory is not reserved in the RA Solution it is possible unrelated data will be linked in this section and will cause conflicts at runtime.

### Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

### Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the data flash erase size of the MCU. It must be greater than 104 bytes which is the minimum block size of a LittleFS block. For information about data flash erase sizes refer to the "Specifications of the code flash memory and data flash memory" table of the "Flash Memory" chapter's "Overview" section.

## Examples

### Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_FLASH_Open(&g_rm_littlefs_flash0_ctrl,
&g_rm_littlefs_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the filesystem. */
```

```
 int lfs_err = lfs_format(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /**********************************************************************************
***********************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
RA Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  **********************************************************************************
*********************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
                               &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_WRONLY |
LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
```

```
/* Apply butter to toast 10 times. */
for (uint32_t i = 0; i < 10; i++)
   {
      lfs_err = lfs_file_write(&g_rm_littlefs_flash0_lfs, &file, contents, len);
if (lfs_err < 0)
     {
     handle_lfs_error(lfs_err);
      }
   }
/* Close the file. */
   lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
   handle_lfs_error(lfs_err);
/* Unmount the filesystem. */
   lfs_err = lfs_unmount(&g_rm_littlefs_flash0_lfs);
   handle_lfs_error(lfs_err);
/* Remount the filesystem. */
   lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
   handle_lfs_error(lfs_err);
/* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
   lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY,
&g_file_cfg);
   handle_lfs_error(lfs_err);
#else
   lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY);
   handle_lfs_error(lfs_err);
#endif
   handle_lfs_error(lfs_err);
/* Verify the toast is buttered the correct amount. */
for (uint32_t i = 0; i < 10; i++)
   {
      lfs_err = lfs_file_read(&g_rm_littlefs_flash0_lfs, &file, buffer, len);
if (lfs_err < 0)
     {
```

```
      handle_lfs_error(lfs_err);
       }
       assert(0 == memcmp(buffer, contents, len));
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
}
```

### Function Documentation

#### ◆ RM_LITTLEFS_FLASH_Open()

fsp_err_t RM_LITTLEFS_FLASH_Open ( rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg* )

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_SIZE | The provided block size is invalid. |
| FSP_ERR_INVALID_ARGUMENT | Flash BGO mode must be disabled. |
| FSP_ERR_INTERNAL | Failed to create the semaphore. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- flash_api_t::open

### ◆ RM_LITTLEFS_FLASH_Close()

fsp_err_t RM_LITTLEFS_FLASH_Close ( rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- flash_api_t::close

#### 5.2.17.13 LittleFS on SPI Flash (rm_littlefs_spi_flash)

Modules » Storage

**Functions**

| | |
|---|---|
| fsp_err_t | RM_LITTLEFS_SPI_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
| fsp_err_t | RM_LITTLEFS_SPI_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl) |

**Detailed Description**

Middleware for the LittleFS File System control on RA MCUs using external SPI Flash.

# Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system:
https://github.com/ARMmbed/littlefs

# Configuration

### Build Time Configurations for rm_littlefs_spi_flash

The following build time configurations are defined in fsp_cfg/rm_littlefs_spi_flash_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for Storage > LittleFS on SPI Flash (rm_littlefs_spi_flash)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rm_littlefs0 | Module name. |
| Delay Callback | Name must be a valid C symbol | g_rm_littlefs_spi_flash0_callback | Optional delay callback for custom delay while waiting for the flash erase/write operation to complete. |
| Read Size | Must be a non-negative integer | 1 | Minimum size of a block read. All read operations will be a multiple of this value. |
| Program Size | Must be a non-negative integer | 4 | Minimum size of a block program. All program operations will be a multiple of this value. |
| Block Size (bytes) | Must be a multiple of 64 | 4096 | Size of an erasable block. This does not impact RAM consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes. |
| Block Cycles | Must be an integer | 1024 | Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling. |
| Cache Size | Must be a non-negative integer | 64 | Size of block caches. Each cache buffers a |

| | | | |
|---|---|---|---|
| | | | portion of a block in RAM. The LittleFS needs a read cache, a program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size. |
| Lookahead Size | Must be a non-negative multiple of 8 | 16 | Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number of blocks found during an allocation pass. The lookahead buffer is stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8. |
| Memory Start Address Offset (bytes) | Offset should be greater than or equal to zero | 0 | Enter the starting offset to use in the SPI memory. The starting address for LittleFS memory will be the SPI memory base address plus this offset. |
| Memory Size (bytes) | Size should be greater than zero | 33554432 | Enter the size that the LittleFS Memory should be. This can be smaller than the SPI memory size in order to use a subset of SPI memory. |
| Poll Status Count | Poll Status Count should be greater than or equal to zero | 0xFFFFFFFF | Number of times to poll for operation complete status for blocking memory operations. |

### Common LittleFS Configuration

### Build Time Configurations for LittleFS

The following build time configurations are defined in arm/littlefs/lfs_util.h:

| Configuration | Options | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Custom lfs_util.h | Manual Entry | | Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations. |
| Thread Safe | • Enabled<br>• Disabled | Disabled | Enables thread safety in LittleFS. |
| Read Only | • Enabled<br>• Disabled | Disabled | Enables Read Only mode in LittleFS. |
| Use Malloc | • Enabled<br>• Disabled | Enabled | Configures the use of malloc by LittleFS. |
| Use Assert | • Enabled<br>• Disabled | Enabled | Configures the use of assert by LittleFS. |
| Debug Messages | • Enabled<br>• Disabled | Disabled | Configures debug messages. |
| Warning Messages | • Enabled<br>• Disabled | Disabled | Configures warning messages. |
| Error Messages | • Enabled<br>• Disabled | Disabled | Configures error messages. |
| Trace Messages | • Enabled<br>• Disabled | Disabled | Configures trace messages. |
| Intrinsics | • Enabled<br>• Disabled | Enabled | Configures intrinsic functions such as __builtin_clz. |
| Instance Name for STDIO wrapper | Name must be a valid C symbol | g_rm_littlefs0 | The rm_littlefs instance name to use with the STDIO wrapper. |

# Usage Notes

### Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

### Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the sector erase size of the SPI Flash. It must be greater than 104 bytes which is the minimum block size of a LittleFS block.

# Examples

### Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_spi_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_SPI_FLASH_Open(&g_rm_littlefs_spi_flash0_ctrl,
&g_rm_littlefs_spi_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the filesystem. */
 int lfs_err = lfs_format(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_spi_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /**********************************************************************************
**********************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
```

```
RA Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  ********************************************************************************
*********************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_spi_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
                               &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_spi_flash0_lfs, &file, path, LFS_O_WRONLY
| LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
 /* Apply butter to toast 10 times. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_write(&g_rm_littlefs_spi_flash0_lfs, &file, contents, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_spi_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
 /* Unmount the filesystem. */
    lfs_err = lfs_unmount(&g_rm_littlefs_spi_flash0_lfs);
```

```
    handle_lfs_error(lfs_err);
 /* Remount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_spi_flash0_lfs,
&g_rm_littlefs_spi_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_spi_flash0_lfs, &file, path,
LFS_O_RDONLY, &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_spi_flash0_lfs, &file, path,
LFS_O_RDONLY);
    handle_lfs_error(lfs_err);
#endif
    handle_lfs_error(lfs_err);
 /* Verify the toast is buttered the correct amount. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_read(&g_rm_littlefs_spi_flash0_lfs, &file, buffer, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
       assert(0 == memcmp(buffer, contents, len));
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_spi_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
}
```

## Function Documentation

### ◆ RM_LITTLEFS_SPI_FLASH_Open()

fsp_err_t RM_LITTLEFS_SPI_FLASH_Open ( rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg* )

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Success. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_SIZE | The provided block size is invalid. |
| FSP_ERR_INVALID_ARGUMENT | Flash BGO mode must be disabled. |
| FSP_ERR_INTERNAL | Failed to create the semaphore. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- spi_flash_api_t::open

### ◆ RM_LITTLEFS_SPI_FLASH_Close()

fsp_err_t RM_LITTLEFS_SPI_FLASH_Close ( rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Media device closed. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | Module not open. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- spi_flash_api_t::close

## 5.2.17.14 MRAM (r_mram)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_MRAM_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_MRAM_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t mram_address, uint32_t const num_bytes) |
| fsp_err_t | R_MRAM_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks) |
| fsp_err_t | R_MRAM_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *p_blank_check_result) |
| fsp_err_t | R_MRAM_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status) |
| fsp_err_t | R_MRAM_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode) |
| fsp_err_t | R_MRAM_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| fsp_err_t | R_MRAM_AccessWindowClear (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MRAM_Reset (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MRAM_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| fsp_err_t | R_MRAM_BankSwap (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MRAM_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MRAM_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info) |
| fsp_err_t | R_MRAM_Close (flash_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_MRAM_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_MRAM_AntiRollbackCounterIncrement (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_MRAM_AntiRollbackCounterRefresh (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter) |
| fsp_err_t | R_MRAM_AntiRollbackCounterRead (flash_ctrl_t *const p_api_ctrl, flash_arc_t counter, uint32_t *const p_count) |

| | |
|---|---|
| fsp_err_t | R_MRAM_UserLockableAreaWrite (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t mram_address, uint32_t const num_bytes) |

### Detailed Description

Driver for the MRAM memory on RA MCUs. This module implements the Flash Interface.

# Overview

The MRAM HAL module APIs allow an application to write and erase MRAM areas that reside within the MCU. The amount of MRAM memory available varies across MCU parts.

### Features

The R_MRAM module has the following key features:

- Blocking writing and erasing of MRAM.
- ECC Error Detection
- Boot block-swapping.
- Anti-rollback counter support (select MCUs).

# Configuration

### Build Time Configurations for r_mram

The following build time configurations are defined in fsp_cfg/r_mram_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Storage > MRAM (r_mram)

This module can be added to the Stacks tab via New Stack > Storage > MRAM (r_mram). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_mram0 | Module name. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified. Callback function called when an ECC Detection Event occurs. |
| MRAM Error Interrupt | MCU Specific Options | | Select the MRAM error |

Priority interrupt priority.

### Clock Configuration

MRAM uses MRAM bus clock for Code MRAM and MRAM clock for Extra MRAM.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Code MRAM Precautions

Code MRAM cannot be accessed while writing or erasing code MRAM. In order to support modifying code MRAM all supporting code must reside in RAM. Code MRAM operation will not return before the operation has completed. By default, the vector table resides in the code MRAM. If an interrupt occurs during the code MRAM operation, then code MRAM will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code MRAM operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must ensure that if in a multi-threaded environment, threads running from code MRAM cannot become active while a code MRAM operation is in progress.

### Erase condsiderations

The erase API was designed with flash blocks in mind. The Erase API provides a mechanism for quickly overwriting blocks of data at a single time. The block size is determined by the MRAM Unit of Programming size.

### Anti Rollback Counter

Anti-rollback counters are available on select MCUs and can be used to prevent or limit firmware downgrade. The counters can only be incremented and never decremented or reset, even with an external programming tool.

*Note*

***The OEM bootloader anti-rollback counter is protected by the security engine and cannot be incremented past the version number specified in the code certificate. This counter works with the First Stage Bootloader (FSBL) set to enabled. For more information, read the 'Secure Boot' section in the 'Security Features' chapter of the user manual of the target MCU and refer to Application Note R11AN0774.***

### Anti-Rollback Counter Configuration Setting

When anti-rollback counters are used for one or more non-secure applications, the Anti-Rollback Counter Configuration Setting for ARC_NSEC (ARCCS) must be configured as either a single line (1xn bits) or multi-line counter (mxn bits). Single line is suitable when there is only one non-secure application using the anti-rollback counter. Multi-line is suitable when there are multiple non-secure applications. Before the counter is configured, it can't be used.

This configuration setting is permanent and must be performed with an external programming tool such as Renesas Flash Programmer (RFP). It cannot be configured in e2 studio.

The secure application and OEM bootloader anti-rollback counters do not require any initial configuration.

### Anti-Rollback Counter Lock Setting

Select MCUs have the ability to permanently lock the anti-rollback counters using the Anti-Rollback Counter Lock Setting (ARCLS). It cannot be set with e2 studio, but if needed, it can be configured with an external programming tool such as Renesas Flash Programmer (RFP). **Once the lock protection is applied to a counter, the counter value can never be incremented again.** It is still possible to read the value of a locked counter.

### Viewing MRAM contents in e2 studio

By default, the contents of code MRAM are cached by e$^2$ studio. This means that during a debug session, modifications to these memory regions will not be observed by e$^2$ studio. When debugging applications using e$^2$ studio, disable the "Allow caching of flash contents" option in the debug configuration in order to view modified MRAM contents (Debug Configuration > Debugger > Debug Tool Settings > Allow caching of flash contents).

### Fault Correction and Error Detection

MRAM systems are equipped with ECC circuitry to correct and detect errors. Interrupts can be enabled to detect certain error conditions. If an error condition has occurred at a given address in memory the interrupt will continue to trigger each time it is accessed until the error condition is resolved. The error condition can be resolved by writing data to the MRAM programming size starting at the address given to the callback.

## Examples

### MRAM Basic Example

This is a basic example of writing to MRAM.

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
void r_mram_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the MRAM instance. */
 fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
    assert(FSP_SUCCESS == err);
 /* Disable interrupts to prevent vector table access while MRAM is programming. */
    __disable_irq();
 /* Write 32 bytes to an area of MRAM. */
```

```
    err = R_MRAM_Write(&g_mram_ctrl, (uint32_t) g_src, MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE);
 /* Enable interrupts after MRAM operations are complete. */
    __enable_irq();
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE));
}
```

### Anti-Rollback Counter Basic Example

This example demonstrates basic usage of the anti-rollback counter functions.

```
void r_mram_anti_rollback_counter_example (void)
{
 /* Open the MRAM instance. */
 fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
    uint32_t current_count;
 /* Read the current counter value for the secure application */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_SEC,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the OEM bootloader */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_OEMBL,
&current_count);
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for the non-secure application if counter is
configured as 1xn bits */
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App
    assert(FSP_SUCCESS == err);
 /* Read the current counter value for 4 non-secure applications if counter is
configured as mxn bits */
```

```
    err = R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_0,
&current_count); // NSEC App 0
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_1,
&current_count); // NSEC App 1
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_2,
&current_count); // NSEC App 2
    err |= R_MRAM_AntiRollbackCounterRead(&g_mram_ctrl, FLASH_ARC_NSEC_3,
&current_count); // NSEC App 3
    assert(FSP_SUCCESS == err);
 /* Increment the counter value for the secure application. Caution: The increment
cannot be reversed. */
    err = R_MRAM_AntiRollbackCounterIncrement(&g_mram_ctrl, FLASH_ARC_SEC);
    assert(FSP_SUCCESS == err);
}
```

### MRAM Two-Bit Error Detection and Correction

This is an example of how a two-bit error condition can be resolved.

```
uint32_t mram_example_refresh[BSP_FEATURE_MRAM_PROGRAMMING_SIZE_BYTES];
static void mram_callback (flash_callback_args_t * p_args)
{
 if (FLASH_EVENT_ERR_TWO_BIT == p_args->event)
    {
 /* Disable interrupts to prevent vector table access while MRAM is programming. */
      __disable_irq();
       memcpy(mram_example_refresh, (uint32_t *) p_args->data,
BSP_FEATURE_MRAM_PROGRAMMING_SIZE_BYTES);
 R_MRAM_Write(&g_mram_ctrl,
                    (uint32_t) mram_example_refresh,
                    p_args->data,
                    BSP_FEATURE_MRAM_PROGRAMMING_SIZE_BYTES);
 /* Enable interrupts after MRAM operations are complete. */
      __enable_irq();
    }
```

```
}
```

### Data Structures

| | |
|---|---|
| struct | mram_instance_ctrl_t |

## Data Structure Documentation

### ◆ mram_instance_ctrl_t

| struct mram_instance_ctrl_t |
|---|
| MRAM instance control block. DO NOT INITIALIZE. |

## Function Documentation

### ◆ R_MRAM_Open()

fsp_err_t R_MRAM_Open ( flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg* )

Initializes the MRAM peripheral. Implements flash_api_t::open.

The Open function initializes the MRAM.

Example:

```
/* Open the MRAM instance. */
fsp_err_t err = R_MRAM_Open(&g_mram_ctrl, &g_mram_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ALREADY_OPEN | The MRAM control block is already open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |
| FSP_ERR_IRQ_BSP_DISABLED | Caller is requesting BGO but the MRAM interrupts are not supported. |

### ◆ R_MRAM_Write()

fsp_err_t R_MRAM_Write ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *mram_address*, uint32_t const *num_bytes* )

Writes to the specified MRAM memory area. Implements flash_api_t::write.

Example:

```
/* Disable interrupts to prevent vector table access while MRAM is programming. */
   __disable_irq();
/* Write 32 bytes to an area of MRAM. */
   err = R_MRAM_Write(&g_mram_ctrl, (uint32_t) g_src, MRAM_WRITE_ADDRESS,
MRAM_EXAMPLE_WRITE_SIZE);
/* Enable interrupts after MRAM operations are complete. */
   __enable_irq();
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation successful. |
| FSP_ERR_IN_USE | The MRAM peripheral is busy with a prior on-going transaction. |
| FSP_ERR_NOT_OPEN | The MRAM API is not Open. |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. This may be returned if the requested MRAM area is not blank. |
| FSP_ERR_TIMEOUT | Timed out waiting for write operation to complete. |
| FSP_ERR_INVALID_SIZE | Number of bytes provided was invalid |
| FSP_ERR_INVALID_ADDRESS | Invalid address was input or address not on programming boundary. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_MRAM_Erase()

fsp_err_t R_MRAM_Erase ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks* )

Erases the specified MRAM blocks. Implements flash_api_t::erase by the block_erase_address.

Example:

```
/* Disable interrupts to prevent vector table access while MRAM is programming. */
   __disable_irq();
/* Erase 10 programmable units of MRAM */
fsp_err_t err = R_MRAM_Erase(&g_mram_ctrl, MRAM_ERASE_ADDRESS, 10);
/* Enable interrupts after MRAM operations are complete. */
   __enable_irq();
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_INVALID_BLOCKS | Invalid number of blocks specified |
| FSP_ERR_INVALID_ADDRESS | Invalid address specified. |
| FSP_ERR_IN_USE | Other MRAM operation in progress, or API not initialized |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | The MRAM API is not Open. |
| FSP_ERR_ERASE_FAILED | Status is indicating a Erase error. |
| FSP_ERR_TIMEOUT | Timed out waiting for the MACI to become ready. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |

### ◆ R_MRAM_BlankCheck()

fsp_err_t R_MRAM_BlankCheck ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result* )

Stub function Implements flash_api_t::blankCheck.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Blank Check is not supported on this MCU. |

### ◆ R_MRAM_StatusGet()

fsp_err_t R_MRAM_StatusGet ( flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status* )

Query the MRAM peripheral for its status. Implements flash_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MRAM peripheral is ready to use. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The MRAM API is not Open. |

### ◆ R_MRAM_IdCodeSet()

fsp_err_t R_MRAM_IdCodeSet ( flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode* )

Stub function Implements flash_api_t::idCodeSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | ID Code is not supported on this MCU. |

### ◆ R_MRAM_AccessWindowSet()

fsp_err_t R_MRAM_AccessWindowSet ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr* )

Stub function Implements flash_api_t::accessWindowSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Access window is not supported on this MCU. |

### ◆ R_MRAM_AccessWindowClear()

fsp_err_t R_MRAM_AccessWindowClear ( flash_ctrl_t *const *p_api_ctrl*)

Stub function Implements flash_api_t::accessWindowClear.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Access window is not supported on this MCU. |

### ◆ R_MRAM_Reset()

fsp_err_t R_MRAM_Reset ( flash_ctrl_t *const *p_api_ctrl*)

Reset the MRAM peripheral. Implements flash_api_t::reset.

No attempt is made to check if the MRAM is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | MRAM circuit successfully reset. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the MACI to become ready. |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |

### ◆ R_MRAM_StartUpAreaSelect()

fsp_err_t R_MRAM_StartUpAreaSelect ( flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary* )

Selects which block, Default (Block 0) or Alternate (Block 1), is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Implements flash_api_t::startupAreaSelect.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | MRAM peripheral is busy with a prior operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_PE_FAILURE | Failed to enter or exit MRAM P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the MACI to become ready. |
| FSP_ERR_WRITE_FAILED | Status is indicating a Programming error for the requested operation. |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |

### ◆ R_MRAM_BankSwap()

fsp_err_t R_MRAM_BankSwap ( flash_ctrl_t *const *p_api_ctrl*)

Stub function flash_api_t::bankSwap.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Bank Swap is not supported on this MCU. |

### ◆ R_MRAM_UpdateFlashClockFreq()

fsp_err_t R_MRAM_UpdateFlashClockFreq ( flash_ctrl_t *const *p_api_ctrl*)

Indicate to the MRAM module that the system clock has changed. Implements flash_api_t::updateFlashClockFreq.

This should be the case if the application has changed the system clock.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Start-up area successfully toggled. |
| FSP_ERR_IN_USE | MRAM is busy with an on-going operation. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl |
| FSP_ERR_NOT_OPEN | MRAM API has not yet been opened. |

### ◆ R_MRAM_InfoGet()

fsp_err_t R_MRAM_InfoGet ( flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info* )

Returns the information about the MRAM regions. Implements flash_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful retrieved the request information. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_info. |

### ◆ R_MRAM_Close()

fsp_err_t R_MRAM_Close ( flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent MRAM operations. Implements flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_ctrl or p_cfg. |

### ◆ R_MRAM_CallbackSet()

fsp_err_t R_MRAM_CallbackSet ( flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void *const *p_context*, flash_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements flash_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_MRAM_AntiRollbackCounterIncrement()

fsp_err_t R_MRAM_AntiRollbackCounterIncrement ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Increments the selected anti-rollback counter.

***Note***

*The counter is read internally before increment.*

Implements flash_api_t::antiRollbackCounterIncrement

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter incremented successfully |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_api_ctrl |
| FSP_ERR_NOT_ENABLED | The specified counter has not been configured (configuration is only required for ARC_NSEC) |
| FSP_ERR_OVERFLOW | The counter cannot be incremented because it is already at its max value |
| FSP_ERR_PE_FAILURE | Failed to enter or exit MRAM P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the MACI to become ready. |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |
| FSP_ERR_UNSUPPORTED | Selected anti-rollback counter is not supported on this MCU |

### ◆ R_MRAM_AntiRollbackCounterRefresh()

fsp_err_t R_MRAM_AntiRollbackCounterRefresh ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter* )

Stub function Implements flash_api_t::antiRollbackCounterRefresh

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Anti-Rollback counter refresh is not supported on this MCU |

### ◆ R_MRAM_AntiRollbackCounterRead()

fsp_err_t R_MRAM_AntiRollbackCounterRead ( flash_ctrl_t *const *p_api_ctrl*, flash_arc_t *counter*, uint32_t *const *p_count* )

Reads the selected anti-rollback counter and returns the number of counter bits set.

Implements flash_api_t::antiRollbackCounterRead

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter read successfully into p_count |
| FSP_ERR_NOT_OPEN | The control block is not open. |
| FSP_ERR_ASSERTION | NULL provided for p_api_ctrl or p_count |
| FSP_ERR_NOT_ENABLED | The specified counter has not been configured (configuration is only required for ARC_NSEC) |
| FSP_ERR_PE_FAILURE | Failed to enter or exit MRAM P/E mode. |
| FSP_ERR_TIMEOUT | Timed out waiting for the MACI to become ready. |
| FSP_ERR_CMD_LOCKED | MRAM Application Command Interface is in locked state |

### ◆ R_MRAM_UserLockableAreaWrite()

fsp_err_t R_MRAM_UserLockableAreaWrite ( flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *mram_address*, uint32_t const *num_bytes* )

Stub function Implements flash_api_t::userLockableAreaWrite

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | User Lockable Area is not supported on this MCU |

## 5.2.17.15 OSPI Flash (r_ospi)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_OSPI_Open (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_OSPI_DirectWrite (spi_flash_ctrl_t *const p_ctrl, uint8_t const |

| | |
|---|---|
| | *const p_src, uint32_t const bytes, bool const read_after_write) |
| fsp_err_t | R_OSPI_DirectRead (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_OSPI_DirectTransfer (spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction) |
| fsp_err_t | R_OSPI_XipEnter (spi_flash_ctrl_t *const p_ctrl) |
| fsp_err_t | R_OSPI_XipExit (spi_flash_ctrl_t *const p_ctrl) |
| fsp_err_t | R_OSPI_Write (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count) |
| fsp_err_t | R_OSPI_Erase (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| fsp_err_t | R_OSPI_StatusGet (spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status) |
| fsp_err_t | R_OSPI_BankSet (spi_flash_ctrl_t *const p_ctrl, uint32_t bank) |
| fsp_err_t | R_OSPI_SpiProtocolSet (spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol) |
| fsp_err_t | R_OSPI_AutoCalibrate (spi_flash_ctrl_t *const p_ctrl) |
| fsp_err_t | R_OSPI_Close (spi_flash_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the OSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.

# Overview

The OSPI peripheral interfaces with an external OctaFlash and/or OctaRAM chip(s) to perform data I/O Operations. When both OctaFlash and OctaRAM devices are interfaced, they must be connected to their own chip-select lines. The devices cannot share a single chip-select line.

### Features

The OSPI driver has the following key features to support the **OctaFlash** device:

- Perform data I/O Operation in both SPI and OPI modes
- Can be configured with OctaFlash device on either of the 2 channels
- Memory mapped read access to the OctaFlash
- Programming the OctaFlash device using single continuous write
- Erasing the OctaFlash device

- Sending device specific commands and reading back responses
- Entering and exiting XIP (Single Continuous Read) mode
- 3 byte addressing for SPI
- 4 byte addressing for SPI and OPI
- Auto-calibration for OPI mode (SOPI and DOPI)

The OSPI driver has the following key features to support the **OctaRAM** device:

- Perform data I/O Operation in DOPI mode
- Can be configured with OctaRAM device on either of the 2 channels
- Memory mapped read amd write access to the OctaRAM using single continuous mode
- Sending device specific commands and reading back responses
- Auto-calibration for DOPI mode
- Uninitialized global variables or buffers can be allocated in the OSPI RAM address space.

Additional build-time features:

- Optional (build-time) DMAC support for data transmission when used with OctaFlash.

*Note*

*For OctaFlash, use of DMAC for data transmission is strongly recommended. Without the use of DMAC, due to the high-speed hardware design of the OSPI peripheral, data transmission can be sensitive to timing variance, which could cause software-based memory-mapped operations to fail unexpectedly.*

# Configuration

**OSPI Flash:**

**Build Time Configurations for r_ospi**

The following build time configurations are defined in driver/r_ospi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DMAC Support | • Enable<br>• Disable | Disable | Enable DMAC support for the OSPI module. |

**Configurations for Storage > OSPI Flash (r_ospi)**

This module can be added to the Stacks tab via New Stack > Storage > OSPI Flash (r_ospi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| General > Single Continuous Mode | | | |
| Read Idle Time | Must be an integer greater than 0 with maximum configurable value of 127 | 100 | Specify the read idle time. |

| | | | |
|---|---|---|---|
| Write Idle Time | Must be an integer greater than 0 with maximum configurable value of 127 | 100 | Specify the write idle time. |
| Name | Name must be a valid C symbol | g_ospi0 | Module name. |
| Channel | Channel should be 0 or 1 | 0 | Specify the OSPI chip select line to use. |
| Flash Size | Must be an integer greater than 0 with maximum configurable value of 0x3FFFFFFF | 0x04000000 | Specify the OctaFlash size in bytes. |
| SPI Protocol | • SPI<br>• Single data rate OPI<br>• Dual data rate OPI | SPI | Select the initial SPI protocol. SPI protocol can be changed on the OctaFlash using R_OSPI_DirectTransfer( ). |
| Address Bytes | • 3<br>• 4 | 4 | Select the number of address bytes. |
| OPI Mode | | | |
| OPI Mode > Auto-Calibration | | | |
| Data latching delay | Must be a valid non-negative integer with maximum configurable value of 0xFF | 0x80 | Set this to 0 to enable auto-calibration. 0x80 is the default value calculated at 3.3V and 25°C |
| Auto-Calibration Address | Must be a valid non-negative integer with maximum configurable value of 0xFFFFFFFF | 0x00 | Set the address of the read/write destination to be performed for auto-calibration. |
| OPI Mode > Command Definitions | | | |
| Page Program Command | Must be a 16-bit OSPI Page Program Command under OPI Mode\|Command Definitions | 0x12ED | The command to program a page in OPI mode. |
| Read Command | Must be a 16-bit OSPI Read Command under OPI Mode\|Command Definitions | 0xEC13 | The command to read in SOPI mode (8READ). |
| Dual Read Command | Must be a 16-bit OSPI Dual Read Command under OPI Mode\|Command Definitions | 0xEE11 | The command to read in DOPI mode (8DTRD). |

| | | | |
|---|---|---|---|
| Write Enable Command | Must be a 16-bit OSPI Write Enable Command under OPI Mode\|Command Definitions | 0x06F9 | The command to enable write in OPI mode. |
| Status Command | Must be a 16-bit OSPI Status Command under OPI Mode\|Command Definitions | 0x05FA | The command to query the status of a write or erase command in OPI mode. |
| OPI Mode > OM_DQS Enable Counter | | | |
| SOPI | Must be an integer between 0 and 255 | 9 | OM_DQS enable counter for memory access. Setting for SOPI mode. |
| DOPI | Must be an integer between 0 and 255 | 6 | OM_DQS enable counter for memory access. Setting for DOPI mode. |
| Command Length Bytes | Must be an integer between 1 and 2 | 2 | Command length in bytes |
| Memory Read Dummy Cycles | Must be an integer between 0 and 255 | 10 | Memory read dummy cycles |
| Status Read Dummy Cycles | Must be an integer between 0 and 255 | 4 | Status read dummy cycles |
| DOPI Byte Order | • Byte0, Byte1, Byte2, Byte3<br>• Byte1, Byte0, Byte3, Byte2 | Byte0, Byte1, Byte2, Byte3 | Byte order on the external bus |
| SPI Mode | | | |
| SPI Mode > Command Definitions | | | |
| Page Program Command | Must be a 8-bit OSPI Page Program Command under SPI Mode\|Command Definitions | 0x12 | The command to program a page in SPI mode. |
| Read Command | Must be a 8-bit OSPI Read Command under SPI Mode\|Command Definitions | 0x13 | The command to read in SPI mode. |
| Write Enable Command | Must be a 16-bit OSPI Write Enable Command under SPI Mode\|Command Definitions | 0x06 | The command to enable write in SPI mode. |
| Status Command | Must be a 16-bit OSPI Status Command under | 0x05 | The command to query the status of a write or |

| | | | |
|---|---|---|---|
| | SPI Mode\|Command Definitions | | erase command in SPI mode. |
| Common Command Definitions | | | |
| Sector Erase Command | Must be a value greater than or equal to 0 | 0x21DE | The command to erase a sector. Set Sector Erase Size to 0 if unused. |
| Block Erase Command | Must be a value greater than or equal to 0 | 0xDC23 | The command to erase a block. Set Block Erase Size to 0 if unused. |
| Chip Erase Command | Must be a value greater than or equal to 0 | 0xC738 | The command to erase the entire chip. Set Chip Erase Command to 0 if unused. |
| Write Status Bit | Must be an integer between 0 and 7 | 0 | Which bit contains the write in progress status returned from the Write Status Command. |
| Write Enable Bit | Must be an integer between 0 and 7 | 1 | Which bit contains the write enable status returned from the Write Enable Command. |
| Sector Erase Size | Must be an integer greater than or equal to 0 and less than 4294967296 | 4096 | The sector erase size. Set Sector Erase Size to 0 if Sector Erase is not supported. |
| Block Erase Size | Must be an integer greater than or equal to 0 and less than 4294967296 | 65536 | The block erase size. Set Block Erase Size to 0 if Block Erase is not supported. |
| Chip Select Timing Setting | | | |
| Memory Mapped Read Command Interval | • 2<br>• 5<br>• 7<br>• 9<br>• 11<br>• 13<br>• 15<br>• 17 | 2 | Memory mapped read command execution interval setting in OCTACLK units |
| Memory Mapped Write Command Interval | • 2<br>• 5<br>• 7<br>• 9<br>• 11<br>• 13 | 2 | Memory mapped write command execution interval setting in OCTACLK units |

| | | | |
|---|---|---|---|
| | • 15<br>• 17 | | |
| Command Interval | • 2<br>• 5<br>• 7<br>• 9<br>• 11<br>• 13<br>• 15<br>• 17 | 2 | Command execution interval setting in OCTACLK units |
| Memory Mapped Read Pull-up Timing | • 5 SPI/SOPI<br>• 6 SPI/SOPI<br>• 7 SPI/SOPI, 6.5 DOPI<br>• 8 SPI/SOPI, 7.5 DOPI<br>• 9 SPI/SOPI, 8.5 DOPI | 5 SPI/SOPI | Memory mapped read signal pull-up timing setting in OCTACLK units |
| Memory Mapped Write Pull-up Timing | • 2 SPI/SOPI, 1.5 DOPI<br>• 3 SPI/SOPI, 2.5 DOPI<br>• 4 SPI/SOPI, 3.5 DOPI<br>• 5 SPI/SOPI, 4.5 DOPI<br>• 6 SPI/SOPI, 5.5 DOPI<br>• 7 SPI/SOPI, 6.5 DOPI<br>• 8 SPI/SOPI, 7.5 DOPI<br>• 9 SPI/SOPI, 8.5 DOPI | 2 SPI/SOPI, 1.5 DOPI | Memory mapped write signal pull-up timing setting in OCTACLK units |
| Pull-up Timing | • 5 SPI/SOPI<br>• 6 SPI/SOPI<br>• 7 SPI/SOPI, 6.5 DOPI<br>• 8 SPI/SOPI, 7.5 DOPI<br>• 9 SPI/SOPI, 8.5 DOPI | 5 SPI/SOPI | Signal pull-up timing setting in OCTACLK units |
| Memory Mapped Read Pull-down Timing | • 3 SPI/SOPI, 2.5 DOPI<br>• 4 SPI/SOPI, 3.5 DOPI<br>• 5 SPI/SOPI, 4.5 DOPI | 3 SPI/SOPI, 2.5 DOPI | Memory mapped read signal pull-down timing setting in OCTACLK units |
| Memory Mapped Write Pull-down Timing | • 3 SPI/SOPI, 2.5 DOPI<br>• 4 SPI/SOPI, 3.5 | 3 SPI/SOPI, 2.5 DOPI | Memory mapped write signal pull-down timing setting in OCTACLK |

| | | | |
|---|---|---|---|
| | DOPI<br>• 5 SPI/SOPI, 4.5 DOPI | | units |
| Pull-down Timing | • 3 SPI/SOPI, 2.5 DOPI<br>• 4 SPI/SOPI, 3.5 DOPI<br>• 5 SPI/SOPI, 4.5 DOPI | 3 SPI/SOPI, 2.5 DOPI | Signal pull-down timing setting in OCTACLK units |

*Note*

*The user is expected to modify the command definitions based on the OctaFlash chip and SPI communication mode. The default mode is SPI mode and default erase commands are set for OPI mode based on Macronix OctaFlash MX25LM51245G.*

**OSPI RAM:**

**Build Time Configurations for r_ospi**

The following build time configurations are defined in driver/r_ospi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| DMAC Support | • Enable<br>• Disable | Disable | Enable DMAC support for the OSPI module. |

**Configurations for Storage > OSPI RAM (r_ospi)**

This module can be added to the Stacks tab via New Stack > Storage > OSPI RAM (r_ospi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| General > Single Continuous Mode | | | |
| Read Idle Time | Must be an integer greater than 0 with maximum configurable value of 127 | 127 | Specify the read idle time. |
| Write Idle Time | Must be an integer greater than 0 with maximum configurable value of 127 | 127 | Specify the write idle time. |
| Name | Name must be a valid C symbol | g_ospi_ram0 | Module name. |
| Channel | Channel should be 0 or 1 [Channel 0 recommended] | 0 | Specify the OSPI chip select line to use. |

| | | | |
|---|---|---|---|
| RAM Size | Must be an integer greater than 0 with maximum configurable value of 0x00800000 | 0x00800000 | Specify the OctaRam size in bytes. |
| SPI Protocol | Dual data rate OPI | Dual data rate OPI | Select the initial SPI protocol. OctaRAM only supports DOPI mode. |
| Address Bytes | 4 | 4 | Select the number of address bytes. OctaRAM only supports 4 byte addresses in DOPI mode. |
| Auto-Calibration | | | |
| Data latching delay | Must be a valid non-negative integer with maximum configurable value of 0xFF | 0x80 | Set this to 0 to enable auto-calibration. 0x80 is the default value calculated at 3.3V and 25°C |
| Auto-Calibration Address | Must be a valid non-negative integer with maximum configurable value of 0xFFFFFFFF | 0x00 | Set the address of the read/write destination to be performed for auto-calibration. |
| Command Definitions | | | |
| Memory Read Command | Must be a 16-bit OSPI Dual Read Command under Command Definitions | 0xA000 | The command to read in DOPI mode. |
| Memory Write Command | Must be a 16-bit OSPI Write Command under Command Definitions | 0x2000 | The command to write in DOPI mode. |
| OM_DQS Enable Counter | | | |
| DOPI | Must be an integer between 0 and 255 | 3 | OM_DQS enable counter for memory access. Setting for DOPI mode. |
| Chip Select Timing Setting | | | |
| Memory Mapped Read Command Interval | • 2<br>• 5<br>• 7<br>• 9<br>• 11<br>• 13<br>• 15<br>• 17 | 2 | Memory mapped read command execution interval setting in OCTACLK units |
| Memory Mapped Write Command Interval | • 2<br>• 5<br>• 7 | 2 | Memory mapped write command execution interval setting in |

| | | | |
|---|---|---|---|
| | • 9<br>• 11<br>• 13<br>• 15<br>• 17 | | OCTACLK units |
| Command Interval | • 2<br>• 5<br>• 7<br>• 9<br>• 11<br>• 13<br>• 15<br>• 17 | 2 | Command execution interval setting in OCTACLK units |
| Memory Mapped Read Pull-up Timing | • 6.5 DOPI<br>• 7.5 DOPI<br>• 8.5 DOPI | 6.5 DOPI | Memory mapped read signal pull-up timing setting in OCTACLK units |
| Memory Mapped Write Pull-up Timing | • 1.5 DOPI<br>• 2.5 DOPI<br>• 3.5 DOPI<br>• 4.5 DOPI<br>• 5.5 DOPI<br>• 6.5 DOPI<br>• 7.5 DOPI<br>• 8.5 DOPI | 1.5 DOPI | Memory mapped write signal pull-up timing setting in OCTACLK units |
| Pull-up Timing | • 6.5 DOPI<br>• 7.5 DOPI<br>• 8.5 DOPI | 6.5 DOPI | Signal pull-up timing setting in OCTACLK units |
| Memory Mapped Read Pull-down Timing | • 2.5 DOPI<br>• 3.5 DOPI<br>• 4.5 DOPI | 2.5 DOPI | Memory mapped read signal pull-down timing setting in OCTACLK units |
| Memory Mapped Write Pull-down Timing | • 2.5 DOPI<br>• 3.5 DOPI<br>• 4.5 DOPI | 2.5 DOPI | Memory mapped write signal pull-down timing setting in OCTACLK units |
| Pull-down Timing | • 2.5 DOPI<br>• 3.5 DOPI<br>• 4.5 DOPI | 2.5 DOPI | Signal pull-down timing setting in OCTACLK units |
| Command Length Bytes | Must be an integer between 1 and 2 | 2 | Command length in bytes |
| Memory Read Dummy Cycles | Must be an integer between 0 and 255 | 4 | Memory read dummy cycles |
| Memory Write Dummy Cycles | Must be an integer between 0 and 255 | 4 | Memory write dummy cycles |
| DOPI Byte Order | • Byte0, Byte1, Byte2, Byte3<br>• Byte1, Byte0, | Byte1, Byte0, Byte3, Byte2 | Byte order on the external bus |

| | | | |
|---|---|---|---|
| | Byte3, Byte2 | | |
| Chip Select Maximum Low Time (us) | Must be an integer between 0 to 511 | 4 | Chip Select Maximum Low Time (tCSM). |

### Clock Configuration

PCLKA is the Octal-SPI bus interface, and PCLKB is used to set OSPI registers.

The signals to the OSPI device are derived from OCTACLK. The OMSCLK signal is OCTACLK / 2. Data can be output at the OCTACLK rate if SPI Protocol is set to Dual Data Rate OPI.

The PCLKB, PCLKA, and OCTACLK frequencies can be set on the **Clocks** tab of the RA Configuration editor.

### Pin Configuration

The following pins are available to connect to an external OSPI device:

- OMSCLK: OSPI clock output (OCTACLK / 2)
- OMDQS: OSPI data strobe signal
- OMCS0: OSPI device 0 select
- OMCS1: OSPI device 1 select
- OMSIO0: Data 0 I/O
- OMSIO1: Data 1 I/O
- OMSIO2: Data 2 I/O
- OMSIO3: Data 3 I/O
- OMSIO4: Data 4 I/O
- OMSIO5: Data 5 I/O
- OMSIO6: Data 6 I/O
- OMSIO7: Data 7 I/O

*Note*

*Data pins must be configured with IOPORT_CFG_DRIVE_HS_HIGH.*
*Chip Select pins should be configured with at least IOPORT_CFG_DRIVE_MEDIUM.*

# Usage Notes

### Usage Notes for OctaFlash support

### Enabling DMAC

DMAC data transmission support is configurable for OSPI Flash and is disabled from the build by default. Use of a high-priority (low channel number) DMAC for data transmission is strongly recommended.

For further details on DMAC please refer Transfer (r_dmac).

### OSPI Memory Mapped Access

After R_OSPI_Open() completes successfully, the OctaFlash device contents are mapped to address 0x68000000 (channel 0) or 0x70000000 (channel 1) based on the channel configured and can be read like on-chip flash. Channel 0 supports 128 MB while Channel 1 supports 256 MB of address space.

### Auto-calibration

Auto-calibration procedure is triggered automatically when the 'Data latching delay' field in the configurator properties is set to 0. The user application is responsible for setting the appropriate preamble pattern before calling R_OSPI_Open() with SOPI/DOPI mode or changing the SPI protocol to SOPI/DOPI using R_OSPI_SpiProtocolSet() API. The appropriate preamble pattern can be written to the desired address using the R_OSPI_Write() API while in the SPI mode (recommended). Ensure that the same address is passed through the configurator. If the OctaFlash chip is already in SOPI/DOPI mode, the preamble pattern must be programmed using the debugger before calling R_OSPI_Open().

### Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support SOPI and SPI at minimum latency. In case the driver is opened in SPI mode and will be switched to DOPI mode later using R_OSPI_SpiProtocolSet(), please select latencies required for DOPI before calling R_OSPI_Open().

### OctaFlash Commands

- Set the erase commands based on intended mode of operation (SPI or OPI). These commands cannot be changed during run-time.
- Read, Write and Status commands for both SPI and OPI are configured allowing switching between the modes at run-time.

## Usage Notes for OctaRAM support

### OSPI Memory Mapped Access

After R_OSPI_Open() completes successfully, the OctaRAM device contents are mapped to address 0x68000000 (channel 0) or 0x70000000 (channel 1) based on the channel configured and can be written to or read from like on-chip RAM. Channel 0 and 1 support 8 MB of address space.

### Auto-calibration

Since the OctaRAM only supports DOPI mode, the driver allows the user to call R_OSPI_Open() without performing the auto-calibration procedure automatically when 'Data latching delay' field is set to 0 in the configurator properties. This is done so that the user can write the appropriate preamble pattern to the desired address using memory mapped writes while in DOPI mode. Ensure that the same address is passed through the configurator. R_OSPI_AutoCalibrate() should be then called to perform auto-calibration.

### Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support DOPI at minimum latency.

### Limitations

Developers should be aware of the following limitations when using the OSPI driver:

### OctaFlash

- Single continuous read in SPI mode is not supported by the peripheral. The maximum amount of data that can be read using a single read command is 4-bytes (When doing a 32-bit access).

- Fast Reads would be slower than regular reads as the SPI mode cannot be operated with an OMSCLK greater than 50MHz.

# Examples

### OSPI Flash:

### Basic Example

This is a basic example of minimal use of the OSPI in an application with OctaFlash.

```
#define OSPI_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[OSPI_EXAMPLE_DATA_LENGTH];
/* Place data in the .ospi0_cs1 section to flash it during programming. */
const uint8_t g_src[OSPI_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".ospi0_cs1") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .ospi0_cs1 section to flash it during programming. */
void r_ospi_example_function(void) BSP_PLACE_IN_SECTION(".ospi0_cs1_code")
__attribute__((noinline));
void r_ospi_example_function (void)
{
 /* Add code here. */
}
void r_ospi_basic_example (void)
{
 /* Open the OSPI instancee */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Change SPI to DOPI mode */
    r_ospi_example_spi_to_dopi();
 /* After R_OSPI_Open() and any required device specific intiialization, data can be
read directly from the OSPI flash. */
    memcpy(&g_dest[0], &g_src[0], OSPI_EXAMPLE_DATA_LENGTH);
 /* After R_OSPI_Open() and any required device specific intiialization, functions in
the OSPI flash can be called. */
    r_ospi_example_function();
}
```

### Reading Status Register Example (R_OSPI_DirectTransfer)

This is an example of using R_OSPI_DirectWrite followed by R_OSPI_DirectRead to send the read status register command and read back the status register from the device.

```
#define OSPI_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_ospi_direct_example (void)
{
 spi_flash_direct_transfer_t ospi_test_direct_transfer =
    {
       .command       = OSPI_TEST_READ_STATUS_COMMAND_SPI_MODE,
       .address       = 0U,
       .data          = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
    };
 /* Open the OSPI instance. */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* Write Enable */
    err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Read Status Register */
    ospi_test_direct_transfer.command    = OSPI_TEST_READ_STATUS_COMMAND_SPI_MODE;
    ospi_test_direct_transfer.data_length = 1U;
    err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Check if Write Enable is set */
 if (OSPI_WEN_BIT_MASK != (ospi_test_direct_transfer.data & OSPI_WEN_BIT_MASK))
    {
       __BKPT(0);
    }
```

```
}
```

### Auto-calibration Example (R_OSPI_DirectTransfer, R_OSPI_Write, R_OSPI_SpiProtocolSet)

This is an example of using R_OSPI_SpiProtocolSet to change the operating mode from SPI to SOPI and allow the driver to initiate auto-calibration.

```
#define OSPI_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES (16U)
#define OSPI_EXAMPLE_PREAMBLE_ADDRESS (0x68000000U) /* Device connected to CS0 */
const uint8_t g_preamble_bytes[OSPI_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES] =
{
    0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x08, 0x00, 0x00, 0xF7, 0xFF, 0x00, 0x08,
0xF7, 0x00, 0xF7
};
void ospi_example_wait_until_wip (void)
{
 fsp_err_t          err = FSP_SUCCESS;
 spi_flash_status_t status;
    status.write_in_progress = true;
    uint32_t timeout = UINT32_MAX;
 while ((status.write_in_progress) && (--timeout))
    {
       err = R_OSPI_StatusGet(&g_ospi0_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
 if (0 == timeout)
    {
      assert(FSP_SUCCESS == err);
    }
}
void r_ospi_auto_calibrate_example (void)
{
 /* Open the OSPI instance. */
 /* Set data_latch_delay_clocks to 0x0 to enable auto-calibration */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
```

```
    assert(FSP_SUCCESS == err);
    uint8_t * preamble_pattern_addr = (uint8_t *) OSPI_EXAMPLE_PREAMBLE_ADDRESS;
    err = R_OSPI_Write(&g_ospi0_ctrl, g_preamble_bytes, preamble_pattern_addr,
OSPI_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
 /* Wait until write has been completed */
    ospi_example_wait_until_wip();
 /* Change from SPI to DOPI mode */
    r_ospi_example_spi_to_dopi();
}
```

### Octaclk Update Example (R_OSPI_SpiProtocolSet)

This is an example of using R_BSP_OctaclkUpdate to change the Octal-SPI clock frequency during run time. The OCTACLK frequency must be updated before calling the R_OSPI_SpiProtocolSet with appropriate clock source and divider settings required to be set for the new SPI protocol mode. Ensure that the clock source selected is started.

```
static void ospi_example_change_omclk (void)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
    octaclk_settings.divider      = BSP_CLOCKS_OCTACLK_DIV_2;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
```

### OSPI Data and IAR

When using the IAR compiler, OSPI data must be const qualified to be downloaded by the debugger.

### OSPI RAM:

### Basic Example

This is a basic example of minimal use of the OSPI in an application with OctaRAM.

```
#define OSPI_RAM_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[OSPI_RAM_EXAMPLE_DATA_LENGTH];
```

```
/* Place uninitialized data buffers in the ospi0_cs1_noinit section.
 * Use ospi0_cs1_noinit section if the OctaRAM is configured on channel 1.
 */
uint8_t g_src_1[OSPI_RAM_EXAMPLE_DATA_LENGTH]
BSP_PLACE_IN_SECTION(".ospi0_cs1_noinit");
uint8_t g_src_2[OSPI_RAM_EXAMPLE_DATA_LENGTH]
BSP_PLACE_IN_SECTION(".ospi0_cs1_noinit");
void r_ospi_ram_basic_example (void)
{
 /* Open the OSPI instancee.
  * Ensure valid setting of the 'Data latching delay' field in the configurator.
  * To successfully perform OSPI RAM reads this value must not be 0.
  */
 fsp_err_t err = R_OSPI_Open(&g_ospi_ram0_ctrl, &g_ospi_ram0_cfg);
    assert(FSP_SUCCESS == err);
 /* After R_OSPI_Open() and any required device specific intiialization, data can be
read from or written to directly from the OSPI RAM. */
    memcpy(&g_dest[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
    memcpy(&g_src_2[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
}
```

### Auto-calibration Example (R_OSPI_DirectTransfer, R_OSPI_AutoCalibrate)

This is an example of using R_OSPI_AutoCalibrate to calibrate OSPI peripheral to read data from the OctaRAM device.

```
#define OSPI_RAM_EXAMPLE_PREAMBLE_ADDRESS (0x68000000U) /* Device connected to CS0 */
#define OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MASK (0x00F0U)
#define OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_POS (4U)
#define OSPI_RAM_EXAMPLE_OCTARAM_100MHZ_4CLOCKS_CR_SETTING (1U)
void r_ospi_ram_auto_calibrate_example (void)
{
 /* Open the OSPI instancee */
 fsp_err_t err = R_OSPI_Open(&g_ospi_ram0_ctrl, &g_ospi_ram0_cfg);
    assert(FSP_SUCCESS == err);
```

```
 /* OctaRAM Configuration Register (cr) read and write command definition */
 spi_flash_direct_transfer_t read_cr =
    {
       .command       = 0xC000U,    // NOLINT(readability-magic-numbers)
       .address       = 0x00040000U, // NOLINT(readability-magic-numbers)
       .data          = 0U,
       .command_length = 2U,
       .address_length = 4U,
       .data_length    = 2U,
 /* Dummy Cycles set to the default value specified in the OctaRAM device
Configuration Register */
       .dummy_cycles   = 5U
    };
 spi_flash_direct_transfer_t write_cr =
    {
       .command       = 0x4000U,    // NOLINT(readability-magic-numbers)
       .address       = 0x00040000U, // NOLINT(readability-magic-numbers)
       .data          = 0U,
       .command_length = 2U,
       .address_length = 4U,
       .data_length    = 2U,
       .dummy_cycles   = 0U
    };
 /* Read OctaRAM device Configuration Register */
    err = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &read_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
    uint16_t config_reg = (uint16_t) (((uint16_t) (read_cr.data) &
~OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MASK) |
                                      ((uint16_t)
(OSPI_RAM_EXAMPLE_OCTARAM_100MHZ_4CLOCKS_CR_SETTING <<
                                                        OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENC
Y_COUNTER_POS) &
                                         OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MA
```

```
SK));
 /* Write Configuration Register */
    write_cr.data = (uint32_t) config_reg;
    err          = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &write_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
    read_cr.data = 0;
 /* Set Dummy Clocks to value configured above (4 Clocks) */
    read_cr.dummy_cycles = 4U;
 /* Read Configuration Register */
    err = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &read_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Confirm the intended Configuration Register value */
    assert(config_reg == (uint16_t) (read_cr.data & UINT16_MAX));
 volatile uint32_t * ram_addr = (uint32_t *) OSPI_RAM_EXAMPLE_PREAMBLE_ADDRESS;
 /* Write the auto-calibration preamble pattern for DOPI mode as specified by the
Hardware Manual */
    ram_addr[0] = 0xFFFF0000;          // NOLINT(readability-magic-numbers)
    ram_addr[1] = 0x0800FF00;          // NOLINT(readability-magic-numbers)
    ram_addr[2] = 0xFF0000F7;          // NOLINT(readability-magic-numbers)
    ram_addr[3] = 0x00F708F7;          // NOLINT(readability-magic-numbers)
    err = R_OSPI_AutoCalibrate(&g_ospi_ram0_ctrl);
    assert(FSP_SUCCESS == err);
 /* After Auto-calibration data can be read from or written to directly from the OSPI
RAM. */
    memcpy(&g_dest[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
    memcpy(&g_src_2[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
}
```

## Data Structures

| | |
|---|---|
| struct | ospi_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | ospi_device_number_t |

| | |
|---|---|
| enum | ospi_device_type_t |
| enum | ospi_command_cs_pullup_clocks_t |
| enum | ospi_command_cs_pulldown_clocks_t |
| enum | ospi_dopi_byte_order_t |

## Data Structure Documentation

### ◆ ospi_instance_ctrl_t

| struct ospi_instance_ctrl_t |
|---|
| Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called |

## Enumeration Type Documentation

### ◆ ospi_device_number_t

| enum ospi_device_number_t | |
|---|---|
| Enumerator | |
| OSPI_DEVICE_NUMBER_0 | Device connected to Chip-Select 0. |
| OSPI_DEVICE_NUMBER_1 | Device connected to Chip-Select 1. |

### ◆ ospi_device_type_t

| enum ospi_device_type_t | |
|---|---|
| Enumerator | |
| OSPI_DEVICE_FLASH | Device Memory type OctaFlash. |
| OSPI_DEVICE_RAM | Device Memory type OctaRAM. |

### ◆ ospi_command_cs_pullup_clocks_t

| enum ospi_command_cs_pullup_clocks_t | |
|---|---|
| Enumerator | |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_2 | 1.5 clocks DOPI mode; 2 Clocks all other modes; Unsupported for DOPI Read |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_3 | 2.5 clocks DOPI mode; 3 Clocks all other modes; Unsupported for DOPI Read |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_4 | 3.5 clocks DOPI mode; 4 Clocks all other modes; Unsupported for DOPI Read |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_5 | 4.5 clocks DOPI mode; 5 Clocks all other modes; Unsupported for DOPI Read |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_6 | 5.5 clocks DOPI mode; 6 Clocks all other modes; Unsupported for DOPI Read |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_7 | 6.5 clocks DOPI mode; 7 Clocks all other modes |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_8 | 7.5 clocks DOPI mode; 8 Clocks all other modes |
| OSPI_COMMAND_CS_PULLUP_CLOCKS_9 | 8.5 clocks DOPI mode; 9 Clocks all other modes |

### ◆ ospi_command_cs_pulldown_clocks_t

| enum ospi_command_cs_pulldown_clocks_t | |
|---|---|
| Enumerator | |
| OSPI_COMMAND_CS_PULLDOWN_CLOCKS_3 | 2.5 clocks DOPI mode; 3 Clocks all other modes |
| OSPI_COMMAND_CS_PULLDOWN_CLOCKS_4 | 3.5 clocks DOPI mode; 4 Clocks all other modes |
| OSPI_COMMAND_CS_PULLDOWN_CLOCKS_5 | 4.5 clocks DOPI mode; 5 Clocks all other modes |

### ◆ ospi_dopi_byte_order_t

| enum ospi_dopi_byte_order_t | |
|---|---|
| Enumerator | |
| OSPI_DOPI_BYTE_ORDER_0123 | DOPI byte order byte 0, byte 1, byte 2, byte 3. |
| OSPI_DOPI_BYTE_ORDER_1032 | DOPI byte order byte 1, byte 0, byte 3, byte 2. |

## Function Documentation

### ◆ R_OSPI_Open()

fsp_err_t R_OSPI_Open ( spi_flash_ctrl_t *const *p_ctrl*, spi_flash_cfg_t const *const *p_cfg* )

Open the OSPI driver module. After the driver is open, the OSPI can be accessed like internal flash memory.

Implements spi_flash_api_t::open.

Example:

```
/* Open the OSPI instancee */
fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | The parameter p_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Driver has already been opened with the same p_ctrl. |
| FSP_ERR_CALIBRATE_FAILED | Failed to perform auto-calibrate. |
| FSP_ERR_INVALID_ARGUMENT | Attempting to open the driver with an invalid SPI protocol for OctaRAM. |

### ◆ R_OSPI_DirectWrite()

fsp_err_t R_OSPI_DirectWrite ( spi_flash_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write* )

Writes raw data directly to the OctaFlash. API not supported. Use R_OSPI_DirectTransfer

Implements spi_flash_api_t::directWrite.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by OSPI. |

### ◆ R_OSPI_DirectRead()

fsp_err_t R_OSPI_DirectRead ( spi_flash_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Reads raw data directly from the OctaFlash. API not supported. Use R_OSPI_DirectTransfer.

Implements spi_flash_api_t::directRead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by OSPI. |

### ◆ R_OSPI_DirectTransfer()

fsp_err_t R_OSPI_DirectTransfer ( spi_flash_ctrl_t *const *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction* )

Read/Write raw data directly with the OctaFlash/OctaRAM device.

Implements spi_flash_api_t::directTransfer.

Example:

```
 /* Write Enable */
    err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_OSPI_XipEnter()

fsp_err_t R_OSPI_XipEnter ( spi_flash_ctrl_t *const *p_ctrl*)

Enters Single Continuous Read/Write mode.

Implements spi_flash_api_t::xipEnter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaRAM. |

### ◆ R_OSPI_XipExit()

fsp_err_t R_OSPI_XipExit ( spi_flash_ctrl_t *const *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaRAM. |

### ◆ R_OSPI_Write()

fsp_err_t R_OSPI_Write ( spi_flash_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count* )

Program a page of data to the flash.

Implements spi_flash_api_t::write.

Example:

```
    err = R_OSPI_Write(&g_ospi0_ctrl, g_preamble_bytes, preamble_pattern_addr,
OSPI_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_DEVICE_BUSY | Another Write/Erase transaction is in progress. |
| FSP_ERR_INVALID_SIZE | Write operation crosses page-boundary. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaRAM. |
| FSP_ERR_WRITE_FAILED | The write enable bit was not set. |

### ◆ R_OSPI_Erase()

fsp_err_t R_OSPI_Erase ( spi_flash_ctrl_t *const *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count* )

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The command to erase the flash was executed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_device_address is NULL, byte_count doesn't match an erase size defined in spi_flash_cfg_t, or byte_count is set to 0. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaRAM. |
| FSP_ERR_WRITE_FAILED | The write enable bit was not set. |

### ◆ R_OSPI_StatusGet()

fsp_err_t R_OSPI_StatusGet ( spi_flash_ctrl_t *const *p_ctrl*, spi_flash_status_t *const *p_status* )

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

Example:

```
      err = R_OSPI_StatusGet(&g_ospi0_ctrl, &status);
     assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The write status is in p_status. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_status is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaRAM. |

### ◆ R_OSPI_BankSet()

fsp_err_t R_OSPI_BankSet ( spi_flash_ctrl_t *const *p_ctrl*, uint32_t *bank* )

Selects the bank to access.

Implements spi_flash_api_t::bankSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by OSPI. |

### ◆ R_OSPI_SpiProtocolSet()

fsp_err_t R_OSPI_SpiProtocolSet ( spi_flash_ctrl_t *const *p_ctrl*, spi_flash_protocol_t *spi_protocol* )

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI protocol updated on MCU peripheral. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_CALIBRATE_FAILED | Failed to perform auto-calibrate. |
| FSP_ERR_INVALID_ARGUMENT | Attempting to set an invalid SPI protocol for OctaRAM. |

### ◆ R_OSPI_AutoCalibrate()

fsp_err_t R_OSPI_AutoCalibrate ( spi_flash_ctrl_t *const *p_ctrl*)

Auto-calibrate the OctaRAM device using the preamble pattern.

***Note***

*The preamble pattern must be written to the configured address before calling this API. Implements spi_flash_api_t::autoCalibrate.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI protocol updated on MCU peripheral. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_CALIBRATE_FAILED | Failed to perform auto-calibrate. |
| FSP_ERR_UNSUPPORTED | API not supported by OSPI - OctaFlash. |

### ◆ R_OSPI_Close()

| fsp_err_t R_OSPI_Close ( spi_flash_ctrl_t *const *p_ctrl*) |
|---|
| Close the OSPI driver module.<br><br>Implements spi_flash_api_t::close.<br><br>**Return values**<br><br>FSP_SUCCESS — Configuration was successful.<br>FSP_ERR_ASSERTION — p_instance_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — Driver is not opened. |

## 5.2.17.16 OSPI Flash (r_ospi_b)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_OSPI_B_Open (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_OSPI_B_DirectWrite (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write) |
| fsp_err_t | R_OSPI_B_DirectRead (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_OSPI_B_DirectTransfer (spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction) |
| fsp_err_t | R_OSPI_B_XipEnter (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_OSPI_B_XipExit (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_OSPI_B_Write (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count) |
| fsp_err_t | R_OSPI_B_Erase (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| fsp_err_t | R_OSPI_B_StatusGet (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status) |
| fsp_err_t | R_OSPI_B_BankSet (spi_flash_ctrl_t *p_ctrl, uint32_t bank) |

| | |
|---|---|
| fsp_err_t | R_OSPI_B_SpiProtocolSet (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol) |
| fsp_err_t | R_OSPI_B_Close (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_OSPI_B_AutoCalibrate (spi_flash_ctrl_t *const p_ctrl) |
| fsp_err_t | R_OSPI_B_RowLoad (spi_flash_ctrl_t *const p_ctrl, uint32_t row_index) |
| fsp_err_t | R_OSPI_B_RowStore (spi_flash_ctrl_t *const p_ctrl, uint32_t row_index) |
| fsp_err_t | R_OSPI_B_DOTF_Configure (spi_flash_ctrl_t *const p_ctrl, ospi_b_dotf_cfg_t *const p_dotf_cfg) |

### Detailed Description

Driver for the OSPI_B peripheral on RA MCUs. This module implements the SPI Flash Interface.

# Overview

The OSPI_B peripheral supports xSPI (or OSPI) compatible external memory devices, and it interfaces with these devices to perform data I/O Operations. The OSPI_B peripheral does not support addressable devices, so all connected memory devices must be connected to an individual chip-select pin. Please note that this document will reference the xSPI protocol to which OSPI is a subset. The OSPI_B peripheral is compatible with a variety of xSPI protocol operating modes.

### Features

The OSPI_B driver has the following key features to support the **xSPI** device:

- Perform data I/O Operation
- Direct memory-mapped access to the xSPI device memory up to 256 MB.
- Can be configured with xSPI devices on either of the 2 channels
- Programming the xSPI device using combination write (up to 64 bytes)
- Erasing the xSPI device
- Sending device specific commands and reading back responses of up to 8 bytes
- 1- and 2-byte addressing intended for row-column memories
- 3 byte addressing
- 4 byte addressing
- Auto-calibration
- Decryption-on-the-fly
- Support for standard SPI frames, xSPI Protocol 1.0, xSPI Protocol 2.0, and xSPI Protocol 2.0 Extended frames.

Additional build-time features:

- Optional (build-time) DMAC support for data transmission when used with OSPI_B.
- Optional (build-time) XiP support for entering/exiting XiP mode of the target device.
- Optional (build-time) Data-strobe (DS) auto-calibration support for target devices using the

DS signal.
- Optional (build-time) Decryption on the fly (DOTF)
- Optional (build-time) Support for memory which use row-column addressing.
- Optional (build-time) BSP initialization of SiP Flash.

*Note*

*For OSPI_B, use of DMAC for data transmission is strongly recommended. Without the use of DMAC, due to the high-speed hardware design of the OSPI peripheral, data transmission can be sensitive to timing variance, which could cause software-based memory-mapped operations to fail unexpectedly.*

# Configuration

### OSPI_B Flash:

### Build Time Configurations for r_ospi_b

The following build time configurations are defined in fsp_cfg/r_ospi_b_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Memory-mapping Support | | | |
| Prefetch Function | • Enable<br>• Disable | Enable | Enable prefetch function on memory-mapped reads. |
| Combination Function | Refer to the RA Configuration tool for available options. | 64 Bytes | Enable combination function on memory-mapped writes. |
| XiP Support | • Enable<br>• Disable | Disable | Enable the use of XiP enter and exit codes. |
| Parameter Checking | • Default (BSP)<br>• Enable<br>• Disable | Default (BSP) | If selected code for parameter checking is included in the build. |
| DMAC Support | • Enable<br>• Disable | Disable | Enable DMAC support for the OSPI module. |
| Autocalibration Support | • Enable<br>• Disable | Disable | Enable DS autocalibration for dual-data-rate modes. |
| DOTF Support | • Enable (Protected Mode)<br>• Enable (Compatibility Mode)<br>• Disable | Disable | Enable DOTF support for the OSPI module. |
| Row Addressing Support | • Enable<br>• Disable | Disable | Enable support for xSPI devices which utilize row-column addressing of the memory array. |

### Configurations for Storage > OSPI (r_ospi_b)

This module can be added to the Stacks tab via New Stack > Storage > OSPI (r_ospi_b).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Must be a valid C symbol name. | g_ospi0 | Module name. |
| Unit | MCU Specific Options | | Specify the OSPI peripheral to use. |
| Chip Select | • CS0<br>• CS1 | CS1 | Specify the OSPI chip select line to use. |
| Write Status Bit | • b0<br>• b1<br>• b2<br>• b3<br>• b4<br>• b5<br>• b6<br>• b7 | b0 | Which bit contains the write in progress status returned from the Write Status Command. |
| Write Enable Bit | • b0<br>• b1<br>• b2<br>• b3<br>• b4<br>• b5<br>• b6<br>• b7 | b1 | Which bit contains the write enable status returned from the Write Enable Command. |
| DS Auto-calibration Pattern Address | Must be a valid pointer address. | 0 | Address to the auto-calibration pattern in the target flash memory's address space. |
| Command Sets | | | |
| Command Sets > Custom Table | | | |
| Table Pointer | Must be a valid C symbol name. | | Specify the custom command set table ( ospi_b_xspi_command_set_t[]) to use. If provided, all other command set settings are ignored. |
| Table Length | Must be an integer greater-than or equal to 0. | 0 | Length of the custom command set table. |
| Command Sets > Erase Sizes | | | |
| Sector Erase | Erase size must be an | 4096 | Size of memory region |

| | | | |
|---|---|---|---|
| | integer greater-than or equal-to zero and less than 4,294,967,296 (0xFFFF_FFFF). | | erased by Sector Erase. |
| Block Erase | Erase size must be an integer greater-than or equal-to zero and less than 4,294,967,296 (0xFFFF_FFFF). | 262144 | Size of memory region erased by Block Erase. |
| Command Sets > Initial Mode | | | |
| Command Sets > Initial Mode > Read | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x13 | Read command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Read commands. Set to 0 to disable dummy cycles. |
| Command Sets > Initial Mode > Program | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x12 | Program command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Program commands. Set to 0 to disable dummy cycles. |
| Command Sets > Initial Mode > Row Load | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x0 | Row Load command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Row Load commands. Set to 0 to disable dummy cycles. |
| Command Sets > Initial Mode > Row Store | | | |
| Command Code | Command codes must be positive and fit in the selected command | 0x0 | Row Store command code. Set to 0x00 to disable. |

| | | | |
|---|---|---|---|
| | byte length. | | |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Row Store commands. Set to 0 to disable dummy cycles. |
| Command Sets > Initial Mode > Write Enable | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x06 | Write Enable command code. Set to 0x00 to disable. |
| Command Sets > Initial Mode > Status Read | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x05 | Status Read command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Status Read commands. Set to 0 to disable dummy cycles. |
| Command Sets > Initial Mode > Sector Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x21 | Sector Erase command code. Set to 0x00 to disable. |
| Command Sets > Initial Mode > Block Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0xDC | Block Erase command code. Set to 0x00 to disable. |
| Command Sets > Initial Mode > Chip Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x60 | Chip Erase command code. Set to 0x00 to disable. |
| Protocol Mode | • SPI (1S-1S-1S)<br>• DSPI (1S-2S-2S)<br>• DSPI (2S-2S-2S)<br>• QSPI (1S-4S-4S)<br>• QSPI (4S-4S-4S)<br>• QSPI (4S-4D-4D)<br>• Dual data rate OPI (8D-8D-8D) | SPI (1S-1S-1S) | Signal protocol mode to use for this command set. |

| | | | |
|---|---|---|---|
| Frame Format | • Standard<br>• xSPI Profile 1.0<br>• xSPI Profile 2.0<br>• xSPI Profile 2.0 Extended | Standard | Frame format to use with this command set. |
| Latency Mode | • Fixed<br>• Variable (DQS) | Fixed | Latency mode to use with this command set. |
| Address Length | • 1 byte<br>• 2 bytes<br>• 3 bytes<br>• 4 bytes | 3 bytes | Number of bytes used to address data in a memory page or row. |
| Address MSB Mask | Address mask must be between 0x00 and 0xFF. | 0xF0 | Mask of bits to zero for the most-significant byte of the address stage during memory-mapped operations. |
| Command Code Length | • 1 byte<br>• 2 bytes | 1 byte | Number of bytes used for command codes. |
| Status Register | • No address<br>• 1 byte<br>• 2 bytes<br>• 3 bytes<br>• 4 bytes | No address | Number of bytes used for addressing the status register. |
| Status Register Address | Status register address must be between 0x00 and 0xFFFF_FFFF, inclusive. | 0x0 | Address of the status register that reports Write-in-Progress and Write-Enable flags. |
| Command Sets > High-speed Mode | | | |
| Command Sets > High-speed Mode > Read | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0xEEEE | Read command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 20 | Dummy cycles to use between the address and data phase for Read commands. Set to 0 to disable dummy cycles. |
| Command Sets > High-speed Mode > Program | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x1212 | Program command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer | 0 | Dummy cycles to use between the address |

| | | | |
|---|---|---|---|
| | between 0 and 31. | | and data phase for Program commands. Set to 0 to disable dummy cycles. |
| Command Sets > High-speed Mode > Row Load | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x0 | Row Load command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Row Load commands. Set to 0 to disable dummy cycles. |
| Command Sets > High-speed Mode > Row Store | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x0 | Row Store command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 0 | Dummy cycles to use between the address and data phase for Row Store commands. Set to 0 to disable dummy cycles. |
| Command Sets > High-speed Mode > Write Enable | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x0606 | Write Enable command code. Set to 0x00 to disable. |
| Command Sets > High-speed Mode > Status Read | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x0505 | Status Read command code. Set to 0x00 to disable. |
| Dummy Cycles | Dummy cycle counts must be an integer between 0 and 31. | 3 | Dummy cycles to use between the address and data phase for Status Read commands. Set to 0 to disable dummy cycles. |
| Command Sets > High-speed Mode > Sector Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x2121 | Sector Erase command code. Set to 0x00 to disable. |

| | | | |
|---|---|---|---|
| Command Sets > High-speed Mode > Block Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0xDCDC | Block Erase command code. Set to 0x00 to disable. |
| Command Sets > High-speed Mode > Chip Erase | | | |
| Command Code | Command codes must be positive and fit in the selected command byte length. | 0x6060 | Chip Erase command code. Set to 0x00 to disable. |
| Protocol Mode | • SPI (1S-1S-1S)<br>• DSPI (1S-2S-2S)<br>• DSPI (2S-2S-2S)<br>• QSPI (1S-4S-4S)<br>• QSPI (4S-4S-4S)<br>• QSPI (4S-4D-4D)<br>• Dual data rate OPI (8D-8D-8D) | Dual data rate OPI (8D-8D-8D) | Signal protocol mode to use for this command set. |
| Frame Format | • Standard<br>• xSPI Profile 1.0<br>• xSPI Profile 2.0<br>• xSPI Profile 2.0 Extended | xSPI Profile 1.0 | Frame format to use with this command set. |
| Latency Mode | • Fixed<br>• Variable (DQS) | Fixed | Latency mode to use with this command set. |
| Address Length | • 1 byte<br>• 2 bytes<br>• 3 bytes<br>• 4 bytes | 4 bytes | Number of bytes used to address data in a memory page or row. |
| Address MSB Mask | Address mask must be between 0x00 and 0xFF. | 0xF0 | Mask of bits to zero for the most-significant byte of the address stage during memory-mapped operations. |
| Command Code Length | • 1 byte<br>• 2 bytes | 2 bytes | Number of bytes used for command codes. |
| Status Register | • No address<br>• 1 byte<br>• 2 bytes<br>• 3 bytes<br>• 4 bytes | 4 bytes | Number of bytes used for addressing the status register. |
| Status Register Address | Status register address must be between 0x00 and 0xFFFF_FFFF, inclusive. | 0x00 | Address of the status register that reports Write-in-Progress and Write-Enable flags. |
| Timing Settings | | | |

| | | | |
|---|---|---|---|
| Timing Settings > Chip Select | | | |
| Command Interval | Refer to the RA Configuration tool for available options. | 2 cycles | Minimum clock cycles between two consecutive xSPI frames (CS idle time). |
| Assertion Extension | • No extension<br>• 1 cycle extension | No extension | Signal pull-up timing (CS asserting extension) setting in OCTACLK units. |
| Negation Extension | • No extension<br>• 1 cycle extension | module.driver.ospi_b.pull_down_timing.clocks_5 | Signal pull-down timing (CS negating extension) setting in OCTACLK units. |
| Timing Settings > SDR | | | |
| Drive Timing | • Drive 1/2 cycle before CK<br>• Drive at CK | Drive 1/2 cycle before CK | Data signal timing relative to the rising-edge of the CK signal. |
| CK Sampling Edge | • Falling edge<br>• Rising edge | Falling edge | Data signal timing relative to the rising-edge of the CK signal. |
| Sampling Delay | • None<br>• 1 cycle<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | None | Number of cycles to delay before sampling the data signal. |
| Timing Settings > DDR | | | |
| Sampling Window Extension | • None<br>• 1 cycle<br>• 2 cycles<br>• 3 cycles<br>• 4 cycles<br>• 5 cycles<br>• 6 cycles<br>• 7 cycles | None | Number of cycles to delay before sampling the data signal. |
| DS Latching Delay | Refer to the RA Configuration tool for available options. | Auto-calibrate | Number of delay cells for the data strobe signal. This value must be set to zero if auto-calibration is used. |
| XiP Mode | | | |
| XiP Enter Code | Must be an integer between 0 and 255, inclusive. | 0 | XiP enter code. |
| XiP Exit Code | Must be an integer | 0 | XiP exit code. |

| | | | |
|---|---|---|---|
| | between 0 and 255, inclusive. | | |
| DOTF | | | |
| Name | Must be a valid C symbol name. | g_ospi_dotf | DOTF configuration name. |
| AES Key | Must be a valid C symbol name. | g_ospi_dotf_key | Name of the encryption key variable. |
| AES IV | Must be a valid C symbol name. | g_ospi_dotf_iv | Name of the encryption IV variable. |
| AES Key Length | MCU Specific Options | | AES key length. |
| Key Format | MCU Specific Options | | Key format. |
| Decryption Start Address | Must be an integer between 0x80000000 and 0x9FFFF000, inclusive. | 0x90000000 | OSPI decryption start address |
| Decryption End Address | Must be an integer between 0x80000FFF and 0x9FFFFFFF, inclusive. | 0x90001FFF | OSPI decryption end address |
| Row Addressing | | | |
| Row Address Length | • No address<br>• 1 byte<br>• 2 bytes<br>• 3 bytes<br>• 4 bytes | No address | Number of bytes used for addressing the row index. |

*Note*

*The user is expected to modify the command definitions based on the xSPI chip and SPI communication mode. The default mode is SPI mode and default erase commands are set for DOPI mode based on Infineon S28HS256.*

### Clock Configuration

The signals to the xSPI target device are derived from OCTACLK. OCTACLK must be set to twice the desired OMSCLK signal frequency. This allows data to be latched on either clock edge for Dual-Data-Rate modes (4S-4D-4D or 8D-8D-8D).

Please refer to the Hardware User Manual for the clocks which control the OSPI_B registers. These along with OCTACLK can be set on the **Clocks** tab of the RA Configuration editor.

### Pin Configuration

The following pins are available to connect to an external OSPI device:

- OMSCLK: OSPI clock output (OCTACLK / 2)
- OMDQS: OSPI data strobe signal
- OMCS0: OSPI device 0 select
- OMCS1: OSPI device 1 select
- OMSIO0: Data 0 I/O

- OMSIO1: Data 1 I/O
- OMSIO2: Data 2 I/O
- OMSIO3: Data 3 I/O
- OMSIO4: Data 4 I/O
- OMSIO5: Data 5 I/O
- OMSIO6: Data 6 I/O
- OMSIO7: Data 7 I/O

*Note*

*Data pins must be configured with IOPORT_CFG_DRIVE_HS_HIGH.*
*Chip Select pins should be configured with at least IOPORT_CFG_DRIVE_MEDIUM.*

# Usage Notes

### Usage Notes for xSPI support

After R_OSPI_B_Open() completes successfully, the xSPI device contents are mapped to the appropriate external memory regions for each OSPI_B unit. For example, on the RA8M1, R_OSPI_B0 maps the CS0 device to address 0x80000000 and the CS1 device to address 0x90000000. After opening, the addresses can be accessed like on-chip flash.

Bank CS0 and CS1 support up to 256 MB of address space each depending on the platform.

### Auto-calibration

If support is enabled, auto-calibration procedures are triggered automatically when the 'Data latching delay' field in the configurator properties is set to 0. The user application is responsible for setting the appropriate preamble pattern before calling R_OSPI_B_Open() when using a data strobe (DS) mode or changing the SPI protocol to a DS mode using the R_OSPI_B_SpiProtocolSet() API. The appropriate preamble pattern can be written to the desired address using the R_OSPI_B_Write() API while in simple SPI mode (recommended), or a non-DS mode. Ensure that the same address is passed through the configurator. If the xSPI device is already in a DS mode, the preamble pattern must be programmed using the debugger before calling R_OSPI_B_Open().

The preamble pattern is expected to be { 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x08, 0x00, 0x00, 0xF7, 0xFF, 0x00, 0x08, 0xF7, 0x00, 0xF7 }.

The autocalibration pattern can be added directly to the binary using the linker script. Please see Programming the Auto-calibration Pattern Using the Linker for an example using LLVM.

### Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support SPI at minimum latency. In case the driver is opened in SPI mode and will be switched to DOPI mode later using R_OSPI_B_SpiProtocolSet(), please select latencies required for DOPI before calling R_OSPI_B_Open().

### XiP Support

OSPI_B supports eXecute in Place (XiP) modes of operation. This can be used for read-only memory-mapped accesses to reduce overall read latency by skipping the command sequence in the xSPI transaction. Separate XiP enter and exit codes may be specified for either attached target device. Upon calling R_OSPI_B_XipEnter(), the associated memory region for the target device is switched to read-only mode and the enter code sent to the device. Calling R_OSPI_B_XipExit() will transmit the

exit code and transition the memory region back to read-write access.

Only one flash device should be used after entering XiP mode. Once entered, XiP codes will be transmitted to all attached devices.

### xSPI Commands

Command sets and erase commands may be specified individually for each supported protocol mode. By default, the configurator only supports an alternative command set for "High-Speed Mode" (DOPI 8D-8D-8D) mode. The command sets cannot be changed during run-time, but the appropriate command set will be selected when changing protocol modes. If a command set is not found, an FSP_ERR_INVALID_MODE error code will be returned.

If custom DOPI erase commands are not specified, ensure the erase commands are the appropriate 2-byte DOPI commands. The lower byte will be used for 1-byte command protocols.

### DOTF Support

Decryption-On-The-Fly is configurable for OSPI Flash and is disabled from the build by default.

Using the DOTF feature requires first creating the encrypted blob on the PC and then configuring the DOTF module with the appropriate parameters to allow decryption of the blob once it is programmed into OSPI. Use the Security Key Management Tool (https://www.renesas.com/us/en/software-tool/security-key-management-tool) to create the encrypted blob.

Example: To encrypt a 4096 byte area in a input srec file from 0x90000000 to 0x90000FFF using a 128 AES encryption key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF and IV 00000000000000000000000000000000 use SKMT with the following arguments:

```
skmt.exe /encdotf /keytype "AES-128" /enckey "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
/nonce "00000000000000000000000000000000" /startaddr "90000000" /endaddr "90000FFF"
/prg "input.srec" /incplain /output "encrypted_output.srec"
```

The values for key, iv and decryption area start and end addresses that were used to create the blob using SKMT must be set in the DOTF configuration in FSP.

Make sure that the Key and IV passed into DOTF configuration are 4 byte aligned. This can be done using a compiler alignment attribute as shown below:

```
uint8_t aes_key[] __attribute__((aligned(4))) =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f
};
```

### Supported Key Formats

- When the Compatibility Mode stack is added for the crypto engine, the user can provide either a wrapped or plaintext AES key to the DOTF module.
- When the Protected Mode stack is used, only wrapped keys are supported.

When configuring the DOTF options, make sure that the key format is specified.

*Note*

***For a detailed description of the Crypto Engine operating modes, refer to Application Note R11AN0498.***

### SiP Flash

SiP or System-in-Package devices, can contain an internally bonded OSPI flash memory in the package. The BSP supports startup initialization of SiP flash memories using the R_OSPI_B driver. To enable startup initialization, please select an appropriate stack under Storage > SiP Flash and enable the following properties in the BSP section of the configurator: OSPI > SiP Flash Support and RA Common > Early BSP Initialization. Both of these BSP properties must be enabled for successful SiP flash startup and initialization of other memory regions (e.g. RAM) which load data from the SiP flash.

## Limitations

Developers should be aware of the following limitations when using the OSPI_B driver:

### OSPI_B

- The R_OSPI_B driver expects memory devices to have uniform sector sizes. If a design requires using a memory with non-uniform sectors, the end-user application must appropriately manage sectors.
- Prefetch and combination support for memory-mapped access is applied globally to all slave devices per OSPI_B unit.
- Combination writes are limited to a maximum of 64 bytes. The user should verify the write has completed before attempting to write more bytes.
- Writing to the memory-mapped regions with the CPU is restricted to 64-bit accesses with 8-byte destination alignments. This restriction is not applicable to other bus masters (e.g. DTC or DMAC).
- When using Arm LLVM ensure any read-only (const) data used with R_OSPI_B_Write() is word (4-byte) aligned if the DMAC is not being used. If parameter checking is enabled, the source pointer alignment will be verified for calls to R_OSPI_B_Write().
- Take care to restrict concurrent accesses of the OSPI memory area. Collisions on the bus can occur if other bus masters attempt to write to the OSPI memory area while another master is reading the OSPI memory area.
- When using 8D-8D-8D mode, care should be taken to access on even-aligned addresses. Problems may occur if odd address alignment is used. This restriction applies to all bus masters using OSPI_B.
- DOTF: AES-192 is not supported when the Protected Mode stack is used.
- When using the IAR compiler, OSPI data must be const qualified to be downloaded by the debugger.
- Memories which use row-column addressing must be managed by the end-user application to ensure the appropriate memory row is loaded and stored.
- When using SiP Flash with encrypted data, the encryption key and IV must be const qualified if BSP startup initialization is used. The SiP flash is initialized before the C-runtime

has been initialized and before any RAM variables have been loaded from their respective non-volatile memories.
- When using SiP Flash, secure partitions must be 32K aligned or programming and access may fail.

# Examples

### OSPI Flash:

The following definitions are used across various example snippets below.

```
#define OSPI_B_COMMAND_WRITE_ENABLE_SPI_MODE (0x06U)
#define OSPI_B_COMMAND_WRITE_DISABLE_SPI_MODE (0x04U)
#define OSPI_B_COMMAND_READ_STATUS_SPI_MODE (0x05U)
#define OSPI_B_COMMAND_READ_REGISTER_SPI_MODE (0x65U)
#define OSPI_B_COMMAND_READ_QSPI_MODE (0x0BU)
#define OSPI_B_COMMAND_PROGRAM_QSPI_MODE (0x12U)
#define OSPI_B_COMMAND_WRITE_ENABLE_QSPI_MODE (0x06U)
#define OSPI_B_COMMAND_STATUS_QSPI_MODE (0x05U)
#define OSPI_B_COMMAND_BLOCK_ERASE_DOPI_MODE (0x21DEU)
#define OSPI_B_COMMAND_SECTOR_ERASE_DOPI_MODE (0xDC23U)
#define OSPI_B_COMMAND_CHIP_ERASE_DOPI_MODE (0x609FU)
#define OSPI_B_COMMAND_READ_DOPI_MODE (0xEE11U)
#define OSPI_B_COMMAND_PROGRAM_DOPI_MODE (0x12EDU)
#define OSPI_B_COMMAND_WRITE_ENABLE_DOPI_MODE (0x06F9U)
#define OSPI_B_COMMAND_STATUS_DOPI_MODE (0x05FAU)
#define OSPI_B_WEN_BIT_MASK (0x02U)
#define OSPI_B_WIP_BIT_MASK (0x01U)
#define OSPI_B_BLOCK_ERASE_SIZE (4096U)
#define OSPI_B_SECTOR_ERASE_SIZE (262144U)
#define OSPI_B_EXAMPLE_DESTINATION_ADDRESS ((void *) 0x90000000U) /* Device on CS1 */
```

```
static void ospi_b_change_mode (spi_flash_protocol_t new_protocol)
{
 /* Step 1. Identify current device configuration and reconfigure as needed... */
 /* Step 2. Update clocks... */
 /* Step 3. Transition driver to the new protocol. */
 fsp_err_t err = R_OSPI_B_SpiProtocolSet(&g_ospi0_ctrl, new_protocol);
```

```
    assert(FSP_SUCCESS == err);
 /* Step 4. Verify mode change... */
}
```

### Basic Example

This is a basic example of minimal use of the OSPI in an application with OctaFlash.

```
#define OSPI_B_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[OSPI_B_EXAMPLE_DATA_LENGTH];
/* Place data in the .ospi_flash section to flash it during programming. */
const uint8_t g_src[OSPI_B_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".ospi0_cs1") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .ospi0_cs1_code section to flash it during programming. */
void r_ospi_b_example_function(void) BSP_PLACE_IN_SECTION(".ospi0_cs1_code")
__attribute__((noinline));
void r_ospi_b_example_function (void)
{
 /* Add code here. */
}
void r_ospi_b_basic_example (void)
{
 /* Open the OSPI instance. */
 fsp_err_t err = R_OSPI_B_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Change SPI to DOPI mode */
    ospi_b_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
 /* After R_OSPI_B_Open() and any required device specific initialization, data can
be read directly from the OSPI flash. */
    memcpy(&g_dest[0], &g_src[0], OSPI_B_EXAMPLE_DATA_LENGTH);
 /* After R_OSPI_B_Open() and any required device specific initialization, functions
in the OSPI flash can be called. */
    r_ospi_b_example_function();
}
```

### Reading Status Register Example (R_OSPI_B_DirectTransfer)

This is an example of using R_OSPI_B_DirectWrite followed by R_OSPI_B_DirectRead to send the read status register command and read back the status register from the device.

```
#define OSPI_B_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_ospi_b_direct_example (void)
{
 spi_flash_direct_transfer_t ospi_b_test_direct_transfer =
    {
       .command       = OSPI_B_COMMAND_READ_STATUS_SPI_MODE,
       .address       = 0U,
       .data          = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
    };
 /* Open the OSPI instance. */
 fsp_err_t err = R_OSPI_B_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* Write Enable */
    err = R_OSPI_B_DirectTransfer(&g_ospi0_ctrl, &ospi_b_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Read Status Register */
    ospi_b_test_direct_transfer.command    = OSPI_B_COMMAND_READ_STATUS_SPI_MODE;
    ospi_b_test_direct_transfer.data_length = 1U;
    err = R_OSPI_B_DirectTransfer(&g_ospi0_ctrl, &ospi_b_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Check if Write Enable is set */
 if (OSPI_B_WEN_BIT_MASK != (ospi_b_test_direct_transfer.data & OSPI_B_WEN_BIT_MASK))
    {
       __BKPT(0);
    }
```

```
}
```

### Auto-calibration Example (R_OSPI_B_DirectTransfer, R_OSPI_B_Write, R_OSPI_B_SpiProtocolSet)

This is an example of using R_OSPI_B_SpiProtocolSet to change the operating mode from SPI to DOPI and allow the driver to initiate auto-calibration.

```
#define OSPI_B_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES (16U)
#define OSPI_B_EXAMPLE_PREAMBLE_ADDRESS (0x90000000U) /* Device connected to CS1 */
const uint8_t g_preamble_bytes[OSPI_B_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES] =
{
    0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x08, 0x00, 0x00, 0xF7, 0xFF, 0x00, 0x08,
0xF7, 0x00, 0xF7
};
static void ospi_b_wait_until_wip (void)
{
 fsp_err_t          err = FSP_SUCCESS;
 spi_flash_status_t status;
    status.write_in_progress = true;
    uint32_t timeout = UINT32_MAX;
 while ((status.write_in_progress) && (--timeout))
    {
       err = R_OSPI_B_StatusGet(&g_ospi0_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
 if (0 == timeout)
    {
      assert(FSP_SUCCESS == err);
    }
}
void r_ospi_b_auto_calibrate_example (void)
{
 /* Open the OSPI instance. */
 /* Set data_latch_delay_clocks to 0x0 to enable auto-calibration */
 fsp_err_t err = R_OSPI_B_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
```

```
    assert(FSP_SUCCESS == err);
    uint8_t * preamble_pattern_addr = (uint8_t *) OSPI_B_EXAMPLE_PREAMBLE_ADDRESS;
    err = R_OSPI_B_Write(&g_ospi0_ctrl, g_preamble_bytes, preamble_pattern_addr,
OSPI_B_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
 /* Wait until write has been completed */
    ospi_b_wait_until_wip();
 /* Change from SPI to DOPI mode */
    ospi_b_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
}
```

### Programming the Auto-calibration Pattern Using the Linker

This is an example of an LLVM LLD linker script which places the auto-calibration pattern into the application binary. For the purposes of this example, the pattern is placed in the second 4K subsector of an ISSI IS25WX memory used on RA8 SiP MCUs.

```
INCLUDE memory_regions.lld
INCLUDE fsp_gen.lld
/* Place the autocalibration pattern in the second 4K subsector. */
OSPI_AUTOCAL_ADDR = SIP_FLASH_START + 0x1000;
SECTIONS
{
    __sip_flash_autocal$$ : AT(OSPI_AUTOCAL_ADDR)
    {
       __sip_flash_autocal$$Base = .;
       LONG(0xFFFF0000)
       LONG(0x000800FF)
       LONG(0x00FFF700)
       LONG(0xF700F708)
       __sip_flash_autocal$$Limit = .;
    }> SIP_FLASH
} INSERT BEFORE .sip_flash.endof;
```

### OctaClk Update Example (R_OSPI_B_SpiProtocolSet)

This is an example of using R_BSP_OctaclkUpdate to change the Octal-SPI clock frequency during run time. The OCTACLK frequency must be updated before calling the R_OSPI_B_SpiProtocolSet with appropriate clock source and divider settings required to be set for the new SPI protocol mode. Ensure that the clock source selected is started.

```
/*
 * Changes the OCTACLK divider; assumes PLL2 is being used as the source for OCTACLK.
 * @param divider New clock divider to switch to.
 */
static void ospi_b_change_omclk (bsp_clocks_octaclk_div_t divider)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
    octaclk_settings.divider      = divider;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
```

### Using a Custom xSPI Command Set Example

This is an example of using custom command sets for 8D-8D-8D and 4S-4S-4S protocol modes.

```
/* Custom two-byte erase commands. */
spi_flash_erase_command_t g_2B_erase_commands[] =
{
    {.command = OSPI_B_COMMAND_BLOCK_ERASE_DOPI_MODE, .size = OSPI_B_BLOCK_ERASE_SIZE
},
    {.command = OSPI_B_COMMAND_SECTOR_ERASE_DOPI_MODE, .size =
OSPI_B_SECTOR_ERASE_SIZE     },
    {.command = OSPI_B_COMMAND_CHIP_ERASE_DOPI_MODE, .size =
SPI_FLASH_ERASE_SIZE_CHIP_ERASE},
};
ospi_b_table_t g_2B_erase_table =
{
    .p_table = g_2B_erase_commands,
    .length = (uint8_t) (sizeof(g_2B_erase_commands) / sizeof
```

```
(g_2B_erase_commands[0])),
};
/* Custom command sets. */
ospi_b_xspi_command_set_t g_command_sets[] =
{
 /* 8D-8D-8D example with inverted lower command byte. */
    {
        .protocol             = SPI_FLASH_PROTOCOL_8D_8D_8D,
        .command_bytes       = OSPI_B_COMMAND_BYTES_2,
        .read_command         = OSPI_B_COMMAND_READ_DOPI_MODE,
        .program_command      = OSPI_B_COMMAND_PROGRAM_DOPI_MODE,
        .write_enable_command = OSPI_B_COMMAND_WRITE_ENABLE_DOPI_MODE,
        .status_command       = OSPI_B_COMMAND_STATUS_DOPI_MODE,
        .read_dummy_cycles    = 20U,
        .program_dummy_cycles = 0U,
        .status_dummy_cycles = 3U,
        .p_erase_commands    = &g_2B_erase_table
    },
 /* 4S-4S-4S example with different .read_command and dummy cycles. */
    {
        .protocol             = SPI_FLASH_PROTOCOL_4S_4S_4S,
        .command_bytes       = OSPI_B_COMMAND_BYTES_1,
        .read_command         = OSPI_B_COMMAND_READ_QSPI_MODE,
        .program_command      = OSPI_B_COMMAND_PROGRAM_QSPI_MODE,
        .write_enable_command = OSPI_B_COMMAND_WRITE_ENABLE_QSPI_MODE,
        .status_command       = OSPI_B_COMMAND_STATUS_QSPI_MODE,
        .read_dummy_cycles    = 10U,
        .program_dummy_cycles = 0U,
        .status_dummy_cycles = 1U,
        .p_erase_commands    = NULL, // Use the common erase definitions.
    },
};
void r_ospi_command_table_example (void)
{
```

```
 /* Open the OSPI instance.
  * Specify `g_command_sets` using the configurator. */
 fsp_err_t err = R_OSPI_B_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* Change SPI to DOPI mode */
    ospi_b_change_mode(SPI_FLASH_PROTOCOL_8D_8D_8D);
 /* After R_OSPI_B_Open() and any required device specific initialization, data can
be read directly from the OSPI flash. */
    memcpy(&g_dest[0], &g_src[0], OSPI_B_EXAMPLE_DATA_LENGTH);
 /* After R_OSPI_B_Open() and any required device specific initialization, functions
in the OSPI flash can be called. */
    r_ospi_b_example_function();
 /* Change SPI to QSPI mode */
    ospi_b_change_mode(SPI_FLASH_PROTOCOL_4S_4S_4S);
 /* Read data from an external device, this time in QSPI mode. */
    memcpy(&g_dest[0], &g_src[0], OSPI_B_EXAMPLE_DATA_LENGTH);
 /* Run the example function again in QSPI mode. */
    r_ospi_b_example_function();
}
```

### Change Protocol Mode Example (R_OSPI_B_DirectTransfer, R_OSPI_B_SpiProtocolSet)

This is an example of using R_OSPI_B_DirectTransfer to change the attached flash device to a new protocol mode during run time.

```
/*
 * This example file is only intended for use with Infineon S28HS256 flash devices
and is non-exhaustive.
 * Other flash devices will require modification of command codes, addresses, and/or
command sequences.
 */
#include "r_ospi_b.h"
/**********************************************************************************
*********************************
 * Macro definitions
```

```
 ***********************************************************************************
*********************************/
/* Command code for enabling write operations. */
#define OSPI_B_COMMAND_WRITE_ENABLE_SPI_MODE (0x06U)
/* Command code to query the status register. */
#define OSPI_B_COMMAND_READ_STATUS_SPI_MODE (0x05U)
/* Command code to read a configuration register. */
#define OSPI_B_COMMAND_READ_REGISTER_SPI_MODE (0x65U)
/* Command code to write to a configuration register. */
#define OSPI_B_COMMAND_WRITE_REGISTER_SPI_MODE (0x71U)
/* Location of the write-enable bit in the status register. */
#define OSPI_B_WEN_BIT_MASK (0x02U)
/* Register setting for CFR5 of the Infineon S28HS256 to enable 8D-8D-8D mode. */
#define OSPI_B_DOPI_REGISTER_SETTING (0x43U)
/* Register setting for CFR3 of the Infineon S28HS256 to support faster clock rates
(up to 166 MHz DOPI). */
#define OSPI_B_FAST_CLOCK_REGISTER_SETTING (0x80U)
/* Address of the CFR3 volatile register for the Infineon S28HS256. */
#define OSPI_B_CFR3V_ADDRESS (0x800004)
/* Address of the CFR5 volatile register for the Infineon S28HS256. */
#define OSPI_B_CFR5V_ADDRESS (0x800006)
/***********************************************************************************
*********************************
 * Function prototypes
 ***********************************************************************************
*********************************/
void r_ospi_b_example_spi_to_dopi(ospi_b_instance_ctrl_t * const p_ctrl);
static void ospi_b_write_enable_and_verify(ospi_b_instance_ctrl_t * const p_ctrl,
bool is_8d);
static void ospi_b_change_omclk(bsp_clocks_octaclk_div_t divider);
/*
 * Enables write operations on the target flash and verifies the change by reading
the status register.
 * @param [in] p_ctrl Pointer to the OSPI_B control struct.
```

```
 * @param is_8d Indicates the target is currently configured for 8D-8D-8D protocol
mode.
 */
static void ospi_b_write_enable_and_verify (ospi_b_instance_ctrl_t * const p_ctrl,
bool is_8d)
{
 spi_flash_direct_transfer_t ospi_b_test_direct_transfer =
    {
       .command       = OSPI_B_COMMAND_WRITE_ENABLE_SPI_MODE,
       .address       = 0U,
       .data          = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
    };
 /* Write Enable */
 fsp_err_t err = R_OSPI_B_DirectTransfer(p_ctrl, &ospi_b_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
    ospi_b_test_direct_transfer.command    = OSPI_B_COMMAND_READ_STATUS_SPI_MODE;
    ospi_b_test_direct_transfer.data_length = 1U;
 /* Adjust the dummy cycles if currently operating in 8D-8D-8D mode. */
 if (is_8d)
    {
       ospi_b_test_direct_transfer.dummy_cycles = 5U;
    }
 /* Read Status Register */
    err = R_OSPI_B_DirectTransfer(p_ctrl, &ospi_b_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 if (OSPI_B_WEN_BIT_MASK != (ospi_b_test_direct_transfer.data & OSPI_B_WEN_BIT_MASK))
    {
 /* Verification of the write enable status failed. */
```

```
        __BKPT(0);
    }
}
/*
 * Changes the OCTACLK divider; assumes PLL2 is being used as the source for OCTACLK.
 * @param divider New clock divider to switch to.
 */
static void ospi_b_change_omclk (bsp_clocks_octaclk_div_t divider)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
    octaclk_settings.divider      = divider;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
/*
 * Example function to change from 1S-1S-1S (SPI) to 8D-8D-8D (DOPI) protocol mode.
 *
 * This assumes S28HS256 is in default factory configuration:
 * - 3-byte address lengths
 * - Memory read latency of 8/20 (1S/8S) up to 166 MHz DOPI
 * - Register read latency of 0/3 up to 25 MHz DPOI
 *
 * @param [in] p_ctrl Pointer to the OSPI_B control struct.
 *
 * The OCTACLK source must be within range to support this example as written. Clocks
are intended to be configured
 * as follows to obtain a PLL2P frequency of 240 MHz and OCTACLK starting frequency
of 60 MHz.
 * HOCO (48 MHz) -> PLL2 -> /3 -> x60.0 {960 MHz} -> PLL2P /4 {240 MHz} -> OCTACLK ->
/4 -> {60 MHz}.
 */
void r_ospi_b_example_spi_to_dopi (ospi_b_instance_ctrl_t * const p_ctrl)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Step 1: Enable write access to the registers. */
    ospi_b_write_enable_and_verify(p_ctrl, false);
 /* Step 2: Adjust the dummy cycles for a faster clock later.
  * Changes the dummy cycles for register commands to 1 for SPI and 5 for OSPI.
  */
 spi_flash_direct_transfer_t xfer_adjust_cfr3 =
    {
       .command       = OSPI_B_COMMAND_WRITE_REGISTER_SPI_MODE,
       .address       = OSPI_B_CFR3V_ADDRESS,
       .data          = OSPI_B_FAST_CLOCK_REGISTER_SETTING,
       .command_length = 1U,
       .address_length = 3U,
       .data_length    = 1U,
       .dummy_cycles   = 0U
    };
    err = R_OSPI_B_DirectTransfer(p_ctrl, &xfer_adjust_cfr3,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Step 3: Change the target flash's configuration to use 8D-8D-8D mode. */
 spi_flash_direct_transfer_t xfer_change_mode =
    {
       .command       = OSPI_B_COMMAND_WRITE_REGISTER_SPI_MODE,
       .address       = OSPI_B_CFR5V_ADDRESS,
       .data          = OSPI_B_DOPI_REGISTER_SETTING,
       .command_length = 1U,
       .address_length = 3U,
       .data_length    = 1U,
       .dummy_cycles   = 0U
    };
    err = R_OSPI_B_DirectTransfer(p_ctrl, &xfer_change_mode,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
```

```
 /* Step 4: Speed up the bus by lowering the input divider.
  * If using the suggested clock settings this will increase the bus speed from 60
MHz to 120 MHz. */
    ospi_b_change_omclk(BSP_CLOCKS_OCTACLK_DIV_2);
 /* Step 5: Transition the OSPI peripheral to DOPI mode. This will initiate auto
calibration as MDTR is 0.
  * This must happen after the device has been re-configured to the new mode.
Otherwise, communications will be
  * lost as both MCU and flash device will be in different modes. */
    err = R_OSPI_B_SpiProtocolSet(p_ctrl, SPI_FLASH_PROTOCOL_DOPI);
    assert(FSP_SUCCESS == err);
 /* Step 6: Verify that the chip is in requested DOPI mode. */
 spi_flash_direct_transfer_t xfer_read_register =
    {
       .command       = OSPI_B_COMMAND_READ_REGISTER_SPI_MODE,
       .address       = OSPI_B_CFR5V_ADDRESS,
       .data          = 0,
       .command_length = 1U,
       .address_length = 4U,
       .data_length    = 1U,
       .dummy_cycles   = 5U,
    };
    err = R_OSPI_B_DirectTransfer(p_ctrl, &xfer_read_register,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 if (OSPI_B_DOPI_REGISTER_SETTING != xfer_read_register.data)
    {
       __BKPT(0);
    }
}
```

## Data Structures

| | |
|---|---|
| struct | ospi_b_table_t |
| struct | ospi_b_timing_setting_t |

| | |
|---|---|
| struct | ospi_b_xspi_command_set_t |
| struct | ospi_b_dotf_cfg_t |
| struct | ospi_b_extended_cfg_t |
| struct | ospi_b_instance_ctrl_t |

### Macros

| | |
|---|---|
| #define | OSPI_B_MAX_WRITE_ENABLE_POLLING_LOOPS |

### Enumerations

| | |
|---|---|
| enum | ospi_b_device_number_t |
| enum | ospi_b_command_bytes_t |
| enum | ospi_b_command_interval_clocks_t |
| enum | ospi_b_command_cs_pullup_clocks_t |
| enum | ospi_b_command_cs_pulldown_clocks_t |
| enum | ospi_b_ds_timing_delay_t |
| enum | ospi_b_sdr_drive_timing_t |
| enum | ospi_b_ck_edge_t |
| enum | ospi_b_sdr_sampling_delay_t |
| enum | ospi_b_ddr_sampling_extension_t |
| enum | ospi_b_frame_format_t |
| enum | ospi_b_latency_mode_t |
| enum | ospi_b_prefetch_function_t |
| enum | ospi_b_combination_function_t |
| enum | ospi_b_dotf_aes_key_length_words_t |
| enum | ospi_b_dotf_aes_key_type_t |
| enum | ospi_b_dotf_key_format_t |

## Data Structure Documentation

### ◆ ospi_b_table_t

| struct ospi_b_table_t | | |
|---|---|---|
| Simple array length table structure. | | |
| Data Fields | | |
| void * | p_table | Pointer to the table array. |
| uint8_t | length | Number of entries in the table. |

### ◆ ospi_b_timing_setting_t

| struct ospi_b_timing_setting_t | | |
|---|---|---|
| Fixed timing configuration for bus signals. | | |
| Data Fields | | |
| ospi_b_command_interval_clocks_t | command_to_command_interval | Interval between 2 consecutive commands. |
| ospi_b_command_cs_pullup_clocks_t | cs_pullup_lag | Duration to de-assert CS line after the last command. |
| ospi_b_command_cs_pulldown_clocks_t | cs_pulldown_lead | Duration to assert CS line before the first command. |
| ospi_b_sdr_drive_timing_t | sdr_drive_timing | Data signal timing relative to the rising-edge of the CK signal. |
| ospi_b_ck_edge_t | sdr_sampling_edge | Selects the clock edge to sample the data signal. |
| ospi_b_sdr_sampling_delay_t | sdr_sampling_delay | Number of cycles to delay before sampling the data signal. |
| ospi_b_ddr_sampling_extension_t | ddr_sampling_extension | Number of cycles to extending the data sampling window in DDR mode. |

### ◆ ospi_b_xspi_command_set_t

| struct ospi_b_xspi_command_set_t | | |
|---|---|---|
| Command set used for a protocol mode. | | |
| Data Fields | | |
| spi_flash_protocol_t | protocol | Protocol mode associated with this command set. |
| ospi_b_frame_format_t | frame_format | Frame format to use for this command set. |
| ospi_b_latency_mode_t | latency_mode | Configurable or variable latency, only valid for OSPI_B_FRAME_FORMAT_XSPI_PROFILE_2 and OSPI_B_FRAME_FORMAT_XSPI_PROFILE_2_EXTENDED. |

| | | |
|---|---|---|
| ospi_b_command_bytes_t | command_bytes | Number of command bytes for each command code. |
| spi_flash_address_bytes_t | address_bytes | Number of bytes used during the address phase. |
| uint16_t | read_command | Read command. |
| uint16_t | program_command | Memory program/write command. |
| uint16_t | write_enable_command | Command to enable write or erase, set to 0x00 to ignore. |
| uint16_t | status_command | Command to read the write status, set to 0x00 to ignore. |
| uint16_t | row_load_command | Load a page into the device's internal buffer, set to 0x00 to ignore. |
| uint16_t | row_store_command | Stores the device's internal buffer to the memory page, set to 0x00 to ignore. |
| uint8_t | read_dummy_cycles | Dummy cycles to be inserted for read commands. |
| uint8_t | program_dummy_cycles | Dummy cycles to be inserted for page program commands. |
| uint8_t | status_dummy_cycles | Dummy cycles to be inserted for status read commands. |
| uint8_t | row_load_dummy_cycles | Dummy cycles to be inserted for the page load command. |
| uint8_t | row_store_dummy_cycles | Dummy cycles to be inserted for the page store command. |
| uint8_t | address_msb_mask | Mask of bits to zero when using memory-mapped operations; only applies to the most-significant byte. |
| bool | status_needs_address | Indicates that reading the status register requires an address stage. |
| uint32_t | status_address | Address to use for reading the status register with "busy" and "write-enable" flags. |
| spi_flash_address_bytes_t | status_address_bytes | Number of bytes used for status register addressing. |
| ospi_b_table_t const * | p_erase_commands | List of all erase commands and associated sizes. |

### ◆ ospi_b_dotf_cfg_t

| struct ospi_b_dotf_cfg_t |
|---|
| This structure is used to hold all the DOTF related configuration. |

### ◆ ospi_b_extended_cfg_t

| struct ospi_b_extended_cfg_t | | |
|---|---|---|
| OSPI_B Extended configuration. | | |
| Data Fields | | |
| uint8_t | ospi_b_unit | The OSPI_B unit corresponding to the selected channel. |
| ospi_b_device_number_t | channel | Device number to be used for memory device. |
| ospi_b_timing_setting_t const * | p_timing_settings | Fixed protocol timing settings. |
| ospi_b_table_t const * | p_xspi_command_set | Additional protocol command sets; if additional protocol commands set are not used set this to NULL. |
| ospi_b_ds_timing_delay_t | data_latch_delay_clocks | Delay after assertion of the DS signal where data should be latched. |
| uint8_t * | p_autocalibration_preamble_pattern_addr | OctaFlash memory address holding the preamble pattern. |
| transfer_instance_t const * | p_lower_lvl_transfer | DMA Transfer instance used for data transmission. |

### ◆ ospi_b_instance_ctrl_t

| struct ospi_b_instance_ctrl_t | | |
|---|---|---|
| Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called | | |
| Data Fields | | |
| spi_flash_cfg_t const * | p_cfg | Pointer to initial configuration. |
| uint32_t | open | Whether or not driver is open. |
| spi_flash_protocol_t | spi_protocol | Current OSPI protocol selected. |
| ospi_b_device_number_t | channel | Device number to be used for memory device. |
| uint8_t | ospi_b_unit | OSPI_B instance number. |
| ospi_b_xspi_command_set_t const * | p_cmd_set | Command set for the active protocol mode. |
| R_XSPI0_Type * | p_reg | Address for the OSPI peripheral associated with this channel. |

## Macro Definition Documentation

### ◆ OSPI_B_MAX_WRITE_ENABLE_POLLING_LOOPS

| #define OSPI_B_MAX_WRITE_ENABLE_POLLING_LOOPS |
|---|
| Maximum number of status polling checks after enabling memory writes. |

## Enumeration Type Documentation

### ◆ ospi_b_device_number_t

| enum ospi_b_device_number_t | |
|---|---|
| OSPI Flash chip select | |
| Enumerator | |
| OSPI_B_DEVICE_NUMBER_0 | Device connected to Chip-Select 0. |
| OSPI_B_DEVICE_NUMBER_1 | Device connected to Chip-Select 1. |

### ◆ ospi_b_command_bytes_t

| enum ospi_b_command_bytes_t | |
|---|---|
| OSPI flash number of command code bytes. | |
| Enumerator | |
| OSPI_B_COMMAND_BYTES_1 | Command codes are 1 byte long. |
| OSPI_B_COMMAND_BYTES_2 | Command codes are 2 bytes long. |

### ◆ ospi_b_command_interval_clocks_t

| enum ospi_b_command_interval_clocks_t | |
|---|---|
| OSPI frame to frame interval | |
| Enumerator | |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_1 | 1 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_2 | 2 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_3 | 3 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_4 | 4 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_5 | 5 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_6 | 6 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_7 | 7 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_8 | 8 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_9 | 9 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_10 | 10 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_11 | 11 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_12 | 12 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_13 | 13 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_14 | 14 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_15 | 15 interval clocks |
| OSPI_B_COMMAND_INTERVAL_CLOCKS_16 | 16 interval clocks |

### ◆ ospi_b_command_cs_pullup_clocks_t

| enum ospi_b_command_cs_pullup_clocks_t | |
|---|---|
| OSPI chip select de-assertion duration | |
| Enumerator | |
| OSPI_B_COMMAND_CS_PULLUP_CLOCKS_NO_EXTENSION | CS asserting No extension. |
| OSPI_B_COMMAND_CS_PULLUP_CLOCKS_1 | CS asserting Extend 1 cycle. |

### ◆ ospi_b_command_cs_pulldown_clocks_t

| enum ospi_b_command_cs_pulldown_clocks_t | |
|---|---|
| OSPI chip select assertion duration | |
| Enumerator | |
| OSPI_B_COMMAND_CS_PULLDOWN_CLOCKS_NO_EXTENSION | CS negating No extension. |
| OSPI_B_COMMAND_CS_PULLDOWN_CLOCKS_1 | CS negating Extend 1 cycle. |

### ◆ ospi_b_ds_timing_delay_t

| enum ospi_b_ds_timing_delay_t | |
|---|---|
| OSPI data strobe delay. | |
| Enumerator | |
| OSPI_B_DS_TIMING_DELAY_NONE | Sample without delay. |
| OSPI_B_DS_TIMING_DELAY_1 | Delay sampling by 1 clock cell. |
| OSPI_B_DS_TIMING_DELAY_2 | Delay sampling by 2 clock cells. |
| OSPI_B_DS_TIMING_DELAY_3 | Delay sampling by 3 clock cells. |
| OSPI_B_DS_TIMING_DELAY_4 | Delay sampling by 4 clock cells. |
| OSPI_B_DS_TIMING_DELAY_5 | Delay sampling by 5 clock cells. |
| OSPI_B_DS_TIMING_DELAY_6 | Delay sampling by 6 clock cells. |
| OSPI_B_DS_TIMING_DELAY_7 | Delay sampling by 7 clock cells. |
| OSPI_B_DS_TIMING_DELAY_8 | |

| | |
|---|---|
| | Delay sampling by 8 clock cells. |
| OSPI_B_DS_TIMING_DELAY_9 | Delay sampling by 9 clock cells. |
| OSPI_B_DS_TIMING_DELAY_10 | Delay sampling by 10 clock cells. |
| OSPI_B_DS_TIMING_DELAY_11 | Delay sampling by 11 clock cells. |
| OSPI_B_DS_TIMING_DELAY_12 | Delay sampling by 12 clock cells. |
| OSPI_B_DS_TIMING_DELAY_13 | Delay sampling by 13 clock cells. |
| OSPI_B_DS_TIMING_DELAY_14 | Delay sampling by 14 clock cells. |
| OSPI_B_DS_TIMING_DELAY_15 | Delay sampling by 15 clock cells. |
| OSPI_B_DS_TIMING_DELAY_16 | Delay sampling by 16 clock cells. |
| OSPI_B_DS_TIMING_DELAY_17 | Delay sampling by 17 clock cells. |
| OSPI_B_DS_TIMING_DELAY_18 | Delay sampling by 18 clock cells. |
| OSPI_B_DS_TIMING_DELAY_19 | Delay sampling by 19 clock cells. |
| OSPI_B_DS_TIMING_DELAY_20 | Delay sampling by 20 clock cells. |
| OSPI_B_DS_TIMING_DELAY_21 | Delay sampling by 21 clock cells. |
| OSPI_B_DS_TIMING_DELAY_22 | Delay sampling by 22 clock cells. |
| OSPI_B_DS_TIMING_DELAY_23 | Delay sampling by 23 clock cells. |
| OSPI_B_DS_TIMING_DELAY_24 | Delay sampling by 24 clock cells. |
| OSPI_B_DS_TIMING_DELAY_25 | Delay sampling by 25 clock cells. |
| OSPI_B_DS_TIMING_DELAY_26 | Delay sampling by 26 clock cells. |
| OSPI_B_DS_TIMING_DELAY_27 | Delay sampling by 27 clock cells. |
| OSPI_B_DS_TIMING_DELAY_28 | Delay sampling by 28 clock cells. |
| OSPI_B_DS_TIMING_DELAY_29 | Delay sampling by 29 clock cells. |
| OSPI_B_DS_TIMING_DELAY_30 | Delay sampling by 30 clock cells. |
| OSPI_B_DS_TIMING_DELAY_31 | Delay sampling by 31 clock cells. |

### ◆ ospi_b_sdr_drive_timing_t

| enum ospi_b_sdr_drive_timing_t | |
|---|---|
| OSPI SDR signal drive timing. | |
| Enumerator | |
| OSPI_B_SDR_DRIVE_TIMING_BEFORE_CK | SDR is asserted 1/2 cycle before the rising-edge of CK. |
| OSPI_B_SDR_DRIVE_TIMING_AT_CK | SDR is asserted at the rising-edge of CK. |

### ◆ ospi_b_ck_edge_t

| enum ospi_b_ck_edge_t | |
|---|---|
| Clock edge useed to sample data in SDR mode. | |
| Enumerator | |
| OSPI_B_CK_EDGE_FALLING | Falling-edge of CK signal. |
| OSPI_B_CK_EDGE_RISING | Rising-edge of CK signal. |

### ◆ ospi_b_sdr_sampling_delay_t

| enum ospi_b_sdr_sampling_delay_t | |
|---|---|
| SDR sampling window delay. | |
| Enumerator | |
| OSPI_B_SDR_SAMPLING_DELAY_NONE | No sampling delay. |
| OSPI_B_SDR_SAMPLING_DELAY_1 | Delay sampling by 1 cycle. |
| OSPI_B_SDR_SAMPLING_DELAY_2 | Delay sampling by 2 cycles. |
| OSPI_B_SDR_SAMPLING_DELAY_3 | Delay sampling by 3 cycles. |
| OSPI_B_SDR_SAMPLING_DELAY_4 | Delay sampling by 4 cycles. |
| OSPI_B_SDR_SAMPLING_DELAY_5 | Delay sampling by 5 cycles. |
| OSPI_B_SDR_SAMPLING_DELAY_6 | Delay sampling by 6 cycles. |
| OSPI_B_SDR_SAMPLING_DELAY_7 | Delay sampling by 7 cycles. |

### ◆ ospi_b_ddr_sampling_extension_t

| enum ospi_b_ddr_sampling_extension_t | |
|---|---|
| DDR sampling window extension. | |
| Enumerator | |
| OSPI_B_DDR_SAMPLING_EXTENSION_NONE | No sampling extension. |
| OSPI_B_DDR_SAMPLING_EXTENSION_1 | Sampling extended by 1 cycle. |
| OSPI_B_DDR_SAMPLING_EXTENSION_2 | Sampling extended by 2 cycles. |
| OSPI_B_DDR_SAMPLING_EXTENSION_3 | Sampling extended by 3 cycles. |
| OSPI_B_DDR_SAMPLING_EXTENSION_4 | Sampling extended by 4 cycles. |
| OSPI_B_DDR_SAMPLING_EXTENSION_5 | Sampling extended by 5 cycles. |
| OSPI_B_DDR_SAMPLING_EXTENSION_6 | Sampling extended by 6 cycles. |
| OSPI_B_DDR_SAMPLING_EXTENSION_7 | Sampling extended by 7 cycles. |

### ◆ ospi_b_frame_format_t

| enum ospi_b_frame_format_t | |
|---|---|
| Format of data frames used for communicating with the target device. | |
| Enumerator | |
| OSPI_B_FRAME_FORMAT_STANDARD | Standard frame with command, address, and data phases. |
| OSPI_B_FRAME_FORMAT_XSPI_PROFILE_1 | JEDEC XSPI 8D-8D-8D Profile 1.0 frame. |
| OSPI_B_FRAME_FORMAT_XSPI_PROFILE_2 | JEDEC XSPI 8D-8D-8D Profile 2.0 frame. |
| OSPI_B_FRAME_FORMAT_XSPI_PROFILE_2_EXTENDED | JEDEC XSPI 8D-8D-8D Profile 2.0 extended 6-byte command-address frame, used with HyperRAM. |

### ◆ ospi_b_latency_mode_t

| enum ospi_b_latency_mode_t | |
|---|---|
| Variable or fixed latency selection for flash devices which can notify the host of requiring additional time. | |
| Enumerator | |
| OSPI_B_LATENCY_MODE_FIXED | Latency is fixed to the number of dummy cycles for the command. |
| OSPI_B_LATENCY_MODE_VARIABLE | The flash target signifies additional latency (2x dummy cycles) by asserting the DQS line during the address phase. |

### ◆ ospi_b_prefetch_function_t

| enum ospi_b_prefetch_function_t | |
|---|---|
| Prefetch function settings | |
| Enumerator | |
| OSPI_B_PREFETCH_FUNCTION_DISABLE | Prefetch function disable. |
| OSPI_B_PREFETCH_FUNCTION_ENABLE | Prefetch function enable. |

### ◆ ospi_b_combination_function_t

| enum ospi_b_combination_function_t | |
|---|---|
| Combination function settings | |
| Enumerator | |
| OSPI_B_COMBINATION_FUNCTION_DISABLE | Combination function disable. |
| OSPI_B_COMBINATION_FUNCTION_4BYTE | Combine up to 4 bytes. |
| OSPI_B_COMBINATION_FUNCTION_8BYTE | Combine up to 8 bytes. |
| OSPI_B_COMBINATION_FUNCTION_12BYTE | Combine up to 12 bytes. |
| OSPI_B_COMBINATION_FUNCTION_16BYTE | Combine up to 16 bytes. |
| OSPI_B_COMBINATION_FUNCTION_20BYTE | Combine up to 20 bytes. |
| OSPI_B_COMBINATION_FUNCTION_24BYTE | Combine up to 24 bytes. |
| OSPI_B_COMBINATION_FUNCTION_28BYTE | Combine up to 28 bytes. |
| OSPI_B_COMBINATION_FUNCTION_32BYTE | Combine up to 32 bytes. |
| OSPI_B_COMBINATION_FUNCTION_36BYTE | Combine up to 36 bytes. |
| OSPI_B_COMBINATION_FUNCTION_40BYTE | Combine up to 40 bytes. |
| OSPI_B_COMBINATION_FUNCTION_44BYTE | Combine up to 44 bytes. |
| OSPI_B_COMBINATION_FUNCTION_48BYTE | Combine up to 48 bytes. |
| OSPI_B_COMBINATION_FUNCTION_52BYTE | Combine up to 52 bytes. |
| OSPI_B_COMBINATION_FUNCTION_56BYTE | Combine up to 56 bytes. |
| OSPI_B_COMBINATION_FUNCTION_60BYTE | Combine up to 60 bytes. |
| OSPI_B_COMBINATION_FUNCTION_64BYTE | Combine up to 64 bytes. |
| OSPI_B_COMBINATION_FUNCTION_2BYTE | Combine up to 2 bytes. |

### ◆ ospi_b_dotf_aes_key_length_words_t

| enum ospi_b_dotf_aes_key_length_words_t |
| --- |
| OSPI DOTF AES Key Lengths. |

### ◆ ospi_b_dotf_aes_key_type_t

| enum ospi_b_dotf_aes_key_type_t |
| --- |
| PSPI DOTF AES Type. |

### ◆ ospi_b_dotf_key_format_t

| enum ospi_b_dotf_key_format_t |
| --- |
| OSPI DOTF key Type. |

## Function Documentation

### ◆ R_OSPI_B_Open()

| fsp_err_t R_OSPI_B_Open ( spi_flash_ctrl_t *const *p_ctrl*, spi_flash_cfg_t const *const *p_cfg* ) |
| --- |
| Open the xSPI device. After the driver is open, the xSPI device can be accessed like internal flash memory.<br><br>Implements spi_flash_api_t::open.<br><br>**Return values**<br><br>FSP_SUCCESS — Configuration was successful.<br>FSP_ERR_ASSERTION — The parameter p_ctrl or p_cfg is NULL.<br>FSP_ERR_ALREADY_OPEN — Driver has already been opened with the same p_ctrl.<br>FSP_ERR_CALIBRATE_FAILED — Failed to perform auto-calibrate. |

### ◆ R_OSPI_B_DirectWrite()

fsp_err_t R_OSPI_B_DirectWrite ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write* )

Writes raw data directly to the OctaFlash. API not supported. Use R_OSPI_B_DirectTransfer

Implements spi_flash_api_t::directWrite.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by OSPI. |

### ◆ R_OSPI_B_DirectRead()

fsp_err_t R_OSPI_B_DirectRead ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Reads raw data directly from the OctaFlash. API not supported. Use R_OSPI_B_DirectTransfer.

Implements spi_flash_api_t::directRead.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by OSPI. |

### ◆ R_OSPI_B_DirectTransfer()

fsp_err_t R_OSPI_B_DirectTransfer ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction* )

Read/Write raw data directly with the OctaFlash.

Implements spi_flash_api_t::directTransfer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_OSPI_B_XipEnter()

fsp_err_t R_OSPI_B_XipEnter ( spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | XiP mode was entered successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | XiP support is not enabled. |

### ◆ R_OSPI_B_XipExit()

fsp_err_t R_OSPI_B_XipExit ( spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | XiP mode was entered successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | XiP support is not enabled. |

### ◆ R_OSPI_B_Write()

fsp_err_t R_OSPI_B_Write ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count* )

Program a page of data to the flash.

Implements spi_flash_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_SIZE | Insufficient space remaining in page or write length is not a multiple of CPU access size when not using the DMAC. |
| FSP_ERR_DEVICE_BUSY | Another Write/Erase transaction is in progress. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |
| FSP_ERR_INVALID_ADDRESS | Destination or source is not aligned to CPU access alignment when not using the DMAC. |

### ◆ R_OSPI_B_Erase()

fsp_err_t R_OSPI_B_Erase ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count* )

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The command to erase the flash was executed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_device_address is NULL, byte_count doesn't match an erase size defined in spi_flash_cfg_t, or byte_count is set to 0. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |

### ◆ R_OSPI_B_StatusGet()

fsp_err_t R_OSPI_B_StatusGet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status* )

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The write status is in p_status. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_status is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_OSPI_B_BankSet()

fsp_err_t R_OSPI_B_BankSet ( spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank* )

Selects the bank to access. Use ospi_b_bank_select_t as the bank value.

Implements spi_flash_api_t::bankSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This function is unsupported. |

### ◆ R_OSPI_B_SpiProtocolSet()

fsp_err_t R_OSPI_B_SpiProtocolSet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol* )

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI protocol updated on MPU peripheral. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_CALIBRATE_FAILED | Failed to perform auto-calibrate. |

### ◆ R_OSPI_B_Close()

fsp_err_t R_OSPI_B_Close ( spi_flash_ctrl_t * *p_ctrl*)

Close the OSPI driver module.

Implements spi_flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_OSPI_B_AutoCalibrate()

fsp_err_t R_OSPI_B_AutoCalibrate ( spi_flash_ctrl_t *const *p_ctrl*)

AutoCalibrate the OSPI_B DS signal.

Implements spi_flash_api_t::autoCalibrate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Autocalibration completed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_UNSUPPORTED | Autocalibration support is not enabled. |
| FSP_ERR_CALIBRATE_FAILED | Failed to perform auto-calibrate. |

### ◆ R_OSPI_B_RowLoad()

fsp_err_t R_OSPI_B_RowLoad ( spi_flash_ctrl_t *const *p_ctrl*, uint32_t *row_index* )

Loads a row into the page buffer on devices which support row addressing.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle. |
| | row_index | Index of the row in the OSPI device. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Page transaction succeeded. |
| FSP_ERR_INVALID_ARGUMENT | Invalid argument. |
| FSP_ERR_NOT_OPEN | Driver has not been opened. |
| FSP_ERR_UNSUPPORTED | Row addressing support is not enabled |
| FSP_ERR_DEVICE_BUSY | On going asynchronous operation or the device in in XiP mode. |

### ◆ R_OSPI_B_RowStore()

fsp_err_t R_OSPI_B_RowStore ( spi_flash_ctrl_t *const *p_ctrl*, uint32_t *row_index* )

Stores (programs) the page buffer to a row on devices which support row addressing.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle. |
| | row_index | Index of the row in the OSPI device. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Page transaction succeeded. |
| FSP_ERR_INVALID_ARGUMENT | Invalid argument. |
| FSP_ERR_NOT_OPEN | Driver has not been opened. |
| FSP_ERR_UNSUPPORTED | Row addressing support is not enabled |
| FSP_ERR_DEVICE_BUSY | On going asynchronous operation or the device in in XiP mode. |

### ◆ R_OSPI_B_DOTF_Configure()

fsp_err_t R_OSPI_B_DOTF_Configure ( spi_flash_ctrl_t *const *p_ctrl*, ospi_b_dotf_cfg_t *const *p_dotf_cfg* )

Configure DOTF

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to OSPI specific control structure |
| [in] | p_dotf_cfg | Pointer to DOTF configuration structure |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | DOTF enabled successfully |
| FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL | Key initialization failed. |
| FSP_ERR_CRYPTO_SCE_FAIL | Key wrapping failed. |
| FSP_ERR_INVALID_ARGUMENT | Invalid key type argument. |
| FSP_ERR_UNSUPPORTED | DOTF support is not enabled |

## 5.2.17.17 QSPI (r_qspi)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | R_QSPI_Open (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| fsp_err_t | R_QSPI_DirectWrite (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write) |
| fsp_err_t | R_QSPI_DirectRead (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| fsp_err_t | R_QSPI_DirectTransfer (spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction) |
| fsp_err_t | R_QSPI_XipEnter (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_XipExit (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_Write (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, |

| | |
|---|---|
| | uint8_t *const p_dest, uint32_t byte_count) |
| fsp_err_t | R_QSPI_Erase (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| fsp_err_t | R_QSPI_StatusGet (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status) |
| fsp_err_t | R_QSPI_BankSet (spi_flash_ctrl_t *p_ctrl, uint32_t bank) |
| fsp_err_t | R_QSPI_SpiProtocolSet (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol) |
| fsp_err_t | R_QSPI_AutoCalibrate (spi_flash_ctrl_t *p_ctrl) |
| fsp_err_t | R_QSPI_Close (spi_flash_ctrl_t *p_ctrl) |

### Detailed Description

Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.

# Overview

### Features

The QSPI driver has the following key features:

- Memory mapped read access to the QSPI flash
- Programming the QSPI flash device
- Erasing the QSPI flash device
- Sending device specific commands and reading back responses
- Entering and exiting QPI mode
- Entering and exiting XIP mode
- 3 or 4 byte addressing

# Configuration

### Build Time Configurations for r_qspi

The following build time configurations are defined in driver/r_qspi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Support Multiple Line Program in Extended SPI Mode | • Enabled<br>• Disabled | Disabled | If selected code for programming on multiple lines in extended SPI mode is |

included in the build.

### Configurations for Storage > QSPI (r_qspi)

This module can be added to the Stacks tab via New Stack > Storage > QSPI (r_qspi).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_qspi0 | Module name. |
| SPI Protocol | • Extended SPI<br>• QPI | Extended SPI | Select the initial SPI protocol. SPI protocol can be changed in R_QSPI_Direct(). |
| Address Bytes | • 3<br>• 4<br>• 4 with 4-byte read code | 3 | Select the number of address bytes. Selecting '4 with 4-byte read code' converts the default read code determined in Read Mode to the 4-byte version. If 4-byte mode is selected without using 4-byte commands, the application must issue the EN4B command using R_QSPI_Direct(). |
| Read Mode | • Standard Read<br>• Fast Read<br>• Fast Read Dual Output<br>• Fast Read Dual I/O<br>• Fast Read Quad Output<br>• Fast Read Quad I/O | Fast Read Quad I/O | Select the read mode for memory mapped access. |
| Dummy Clocks for Fast Read | Refer to the RA Configuration tool for available options. | Default | Select the number of dummy clocks for fast read operations. Default is 6 clocks for Fast Read Quad I/O, 4 clocks for Fast Read Dual I/O, and 8 clocks for other fast read instructions including Fast Read Quad Output, Fast Read Dual Output, and Fast Read |

| | | | |
|---|---|---|---|
| Page Size Bytes | Must be an integer greater than 0 | 256 | The maximum number of bytes allowed for a single write. |
| Command Definitions | | | |
| Page Program Command | Must be an 8-bit QSPI Page Program Command under Command Definitions | 0x02 | The command to program a page. If 'Support Multiple Line Program in Extended SPI Mode' is Enabled, this command must use the same number of data lines as the selected read mode. |
| Page Program Address Lines | • 1<br>• 2<br>• 4 | 1 | Select the number of lines to use for the address bytes during write operations. This can be determined by referencing the datasheet for the external QSPI. It should either be 1 or match the number of data lines used for memory mapped fast read operations. |
| Write Enable Command | Must be an 8-bit QSPI Write Enable Command under Command Definitions | 0x06 | The command to enable write. |
| Status Command | Must be an 8-bit QSPI Status Command under Command Definitions | 0x05 | The command to query the status of a write or erase command. |
| Write Status Bit | Must be an integer between 0 and 7 | 0 | Which bit contains the write in progress status returned from the Write Status Command. |
| Sector Erase Command | Must be an 8-bit QSPI Sector Erase Command under Command Definitions | 0x20 | The command to erase a sector. Set Sector Erase Size to 0 if unused. |
| Sector Erase Size | Must be an integer greater than or equal to 0 | 4096 | The sector erase size. Set Sector Erase Size to 0 if Sector Erase is not supported. |
| Block Erase Command | Must be an 8-bit QSPI Block Erase Command under Command | 0xD8 | The command to erase a block. Set Block Erase Size to 0 if |

| | | | |
|---|---|---|---|
| | Definitions | | unused. |
| Block Erase Size | Must be an integer greater than or equal to 0 | 65536 | The block erase size. Set Block Erase Size to 0 if Block Erase is not supported. |
| Block Erase 32KB Command | Must be an 8-bit QSPI Block Erase 32KB Command under Command Definitions | 0x52 | The command to erase a 32KB block. Set Block Erase Size to 0 if unused. |
| Block Erase 32KB Size | Must be an integer greater than or equal to 0 | 32768 | The block erase 32KB size. Set Block Erase 32KB Size to 0 if Block Erase 32KB is not supported. |
| Chip Erase Command | Must be an 8-bit QSPI Chip Erase Command under Command Definitions | 0xC7 | The command to erase the entire chip. Set Chip Erase Command to 0 if unused. |
| XIP Enter M7-M0 | Must be an 8-bit QSPI XIP Enter M7-M0 command under Command Definitions | 0x20 | How to set M7-M0 to enter XIP mode. |
| XIP Exit M7-M0 | Must be an 8-bit QSPI XIP Exit M7-M0 command under Command Definitions | 0xFF | How to set M7-M0 exit XIP mode. |
| Bus Timing | | | |
| QSPKCLK Divisor | MCU Specific Options | | Select the divisor to apply to PCLK to get QSPCLK. |
| Minimum QSSL Deselect Cycles | Refer to the RA Configuration tool for available options. | 4 QSPCLK | Define the minimum number of QSPCLK cycles for QSSL to remain high between operations. |

### Clock Configuration

The QSPI clock is derived from PCLKA.

### Pin Configuration

The following pins are available to connect to an external QSPI device:

- QSPCLK: QSPI clock output
- QSSL: QSPI slave select
- QIO0: Data 0 I/O
- QIO1: Data 1 I/O
- QIO2: Data 2 I/O

- QIO3: Data 3 I/O

*Note*

*It is recommended to configure the pins with IOPORT_CFG_DRIVE_HIGH.*

# Usage Notes

### QSPI Memory Mapped Access

After R_QSPI_Open() completes successfully, the QSPI flash device contents are mapped to address 0x60000000 and can be read like on-chip flash.

### Limitations

Developers should be aware of the following limitations when using the QSPI driver:

- Only P305-P310 are currently supported by the J-Link driver to flash the QSPI.
- The default J-Link downloader requires the device to be in extended SPI mode (not QPI mode).

# Examples

### Basic Example

This is a basic example of minimal use of the QSPI in an application.

```
#define QSPI_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[QSPI_EXAMPLE_DATA_LENGTH];
/* Place data in the .qspi_flash section to flash it during programming. */
const uint8_t g_src[QSPI_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".qspi_flash") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .qspi_flash_code section to flash it during programming. */
void r_qspi_example_function(void) BSP_PLACE_IN_SECTION(".qspi_flash_code")
__attribute__((noinline));
void r_qspi_example_function (void)
{
 /* Add code here. */
}
void r_qspi_basic_example (void)
{
 /* Open the QSPI instance. */
 fsp_err_t err = R_QSPI_Open(&g_qspi0_ctrl, &g_qspi0_cfg);
    assert(FSP_SUCCESS == err);
```

```
 /* (Optional) Send device specific initialization commands. */
    r_qspi_example_init();
 /* After R_QSPI_Open() and any required device specific intiialization, data can be
read directly from the QSPI flash. */
    memcpy(&g_dest[0], &g_src[0], QSPI_EXAMPLE_DATA_LENGTH);
 /* After R_QSPI_Open() and any required device specific intiialization, functions in
the QSPI flash can be called. */
    r_qspi_example_function();
}
```

### Initialization Command Structure Example

This is an example of the types of commands that can be used to initialize the QSPI.

```
#define QSPI_COMMAND_WRITE_ENABLE (0x06U)
#define QSPI_COMMAND_WRITE_STATUS_REGISTER (0x01U)
#define QSPI_COMMAND_ENTER_QPI_MODE (0x38U)
#define QSPI_EXAMPLE_STATUS_REGISTER_1 (0x40)
#define QSPI_EXAMPLE_STATUS_REGISTER_2 (0x00)
static void r_qspi_example_init (void)
{
 /* Write status registers */
 /* Write one byte to enable writing to the status register, then deassert QSSL. */
    uint8_t   data[4];
 fsp_err_t err;
    data[0] = QSPI_COMMAND_WRITE_ENABLE;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    assert(FSP_SUCCESS == err);
 /* Write 3 bytes, including the write status register command followed by values for
both status registers. In the
  * status registers, set QE to 1 and other bits to their default setting. After all
data is written, deassert the
  * QSSL line. */
    data[0] = QSPI_COMMAND_WRITE_STATUS_REGISTER;
    data[1] = QSPI_EXAMPLE_STATUS_REGISTER_1;
```

```
    data[2] = QSPI_EXAMPLE_STATUS_REGISTER_2;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 3, false);
    assert(FSP_SUCCESS == err);
 /* Wait for status register to update. */
 spi_flash_status_t status;
 do
    {
       (void) R_QSPI_StatusGet(&g_qspi0_ctrl, &status);
    } while (true == status.write_in_progress);
 /* Write one byte to enter QSPI mode, then deassert QSSL. After entering QPI mode on
the device, change the SPI
  * protocol to QPI mode on the MCU peripheral. */
    data[0] = QSPI_COMMAND_ENTER_QPI_MODE;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    assert(FSP_SUCCESS == err);
    (void) R_QSPI_SpiProtocolSet(&g_qspi0_ctrl, SPI_FLASH_PROTOCOL_QPI);
}
```

### Reading Status Register Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to send the read status register command and read back the status register from the device.

```
#define QSPI_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_qspi_direct_example (void)
{
 /* Read a status register. */
 /* Write one byte to read the status register. Do not deassert QSSL. */
    uint8_t   data;
 fsp_err_t err;
    data = QSPI_COMMAND_READ_STATUS_REGISTER;
    err = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data, 1, true);
    assert(FSP_SUCCESS == err);
 /* Read one byte. After all data is read, deassert the QSSL line. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data, 1);
```

```
    assert(FSP_SUCCESS == err);
 /* Status register contents are available in variable 'data'. */
}
```

### Querying Device Size Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to query the device size.

```
#define QSPI_EXAMPLE_COMMAND_READ_ID (0x9F)
#define QSPI_EXAMPLE_COMMAND_READ_SFDP (0x5A)
void r_qspi_size_example (void)
{
 /* Many QSPI devices support more than one way to query the device size. Consult the
datasheet for your
  * QSPI device to determine which of these methods are supported (if any). */
    uint32_t device_size_bytes;
 fsp_err_t err;
#ifdef QSPI_EXAMPLE_COMMAND_READ_ID
 /* This example shows how to get the device size by reading the manufacturer ID. */
    uint8_t data[4];
    data[0] = QSPI_EXAMPLE_COMMAND_READ_ID;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, true);
    assert(FSP_SUCCESS == err);
 /* Read 3 bytes. The third byte often represents the size of the QSPI, where the
size of the QSPI = 2 ^ N. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data[0], 3);
    assert(FSP_SUCCESS == err);
    device_size_bytes = 1U << data[2];
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
#ifdef QSPI_EXAMPLE_COMMAND_READ_SFDP
 /* Read the JEDEC SFDP header to locate the JEDEC flash parameters table. Reference
JESD216 "Serial Flash
  * Discoverable Parameters (SFDP)". */
```

```
 /* Send the standard 0x5A command followed by 3 address bytes (SFDP header is at
address 0). */
    uint8_t buffer[16];
    memset(&buffer[0], 0, sizeof(buffer));
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    assert(FSP_SUCCESS == err);
 /* Read out 16 bytes (1 dummy byte followed by 15 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 16);
    assert(FSP_SUCCESS == err);
 /* Read the JEDEC flash parameters to locate the memory size. */
 /* Send the standard 0x5A command followed by 3 address bytes (located in big endian
order at offset 0xC-0xE).
  * These bytes are accessed at 0xD-0xF because the first byte read is a dummy byte.
*/
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    buffer[1] = buffer[0xF];
    buffer[2] = buffer[0xE];
    buffer[3] = buffer[0xD];
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    assert(FSP_SUCCESS == err);
 /* Read out 9 bytes (1 dummy byte followed by 8 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 9);
    assert(FSP_SUCCESS == err);
 /* Read the memory density (located in big endian order at offset 0x4-0x7). These
bytes are accessed at 0x5-0x8
  * because the first byte read is a dummy byte. */
    uint32_t memory_density = (uint32_t) ((buffer[8] << 24) | (buffer[7] << 16) |
(buffer[6] << 8) | buffer[5]);
 if ((1U << 31) & memory_density)
    {
 /* For densities 4 gigabits and above, bit-31 is set to 1b. The field 30:0 defines
'N' where the density is
  * computed as 2^N bits (N must be >= 32). This code subtracts 3 from N to divide by
```

```
8 to get the size in
  * bytes instead of bits. */
       device_size_bytes = 1U << ((memory_density & ~(1U << 31)) - 3U);
    }
 else
    {
 /* For densities 2 gigabits or less, bit-31 is set to 0b. The field 30:0 defines the
size in bits. This
  * code divides the memory density by 8 to get the size in bytes instead of bits. */
       device_size_bytes = (memory_density / 8) + 1;
    }
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
}
```

## Data Structures

| | |
|---|---|
| struct | qspi_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | qspi_qssl_min_high_level_t |
| enum | qspi_qspclk_div_t |

## Data Structure Documentation

### ◆ qspi_instance_ctrl_t

| struct qspi_instance_ctrl_t |
|---|
| Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called |

## Enumeration Type Documentation

### ◆ qspi_qssl_min_high_level_t

| enum qspi_qssl_min_high_level_t | |
|---|---|
| Enumerator | |
| QSPI_QSSL_MIN_HIGH_LEVEL_1_QSPCLK | QSSL deselected for at least 1 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_2_QSPCLK | QSSL deselected for at least 2 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_3_QSPCLK | QSSL deselected for at least 3 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_4_QSPCLK | QSSL deselected for at least 4 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_5_QSPCLK | QSSL deselected for at least 5 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_6_QSPCLK | QSSL deselected for at least 6 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_7_QSPCLK | QSSL deselected for at least 7 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_8_QSPCLK | QSSL deselected for at least 8 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_9_QSPCLK | QSSL deselected for at least 9 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_10_QSPCLK | QSSL deselected for at least 10 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_11_QSPCLK | QSSL deselected for at least 11 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_12_QSPCLK | QSSL deselected for at least 12 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_13_QSPCLK | QSSL deselected for at least 13 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_14_QSPCLK | QSSL deselected for at least 14 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_15_QSPCLK | QSSL deselected for at least 15 QSPCLK. |
| QSPI_QSSL_MIN_HIGH_LEVEL_16_QSPCLK | QSSL deselected for at least 16 QSPCLK. |

### ◆ qspi_qspclk_div_t

| enum qspi_qspclk_div_t | |
|---|---|
| Enumerator | |
| QSPI_QSPCLK_DIV_2 | QSPCLK = PCLK / 2. |
| QSPI_QSPCLK_DIV_3 | QSPCLK = PCLK / 3. |
| QSPI_QSPCLK_DIV_4 | QSPCLK = PCLK / 4. |
| QSPI_QSPCLK_DIV_5 | QSPCLK = PCLK / 5. |
| QSPI_QSPCLK_DIV_6 | QSPCLK = PCLK / 6. |
| QSPI_QSPCLK_DIV_7 | QSPCLK = PCLK / 7. |
| QSPI_QSPCLK_DIV_8 | QSPCLK = PCLK / 8. |
| QSPI_QSPCLK_DIV_9 | QSPCLK = PCLK / 9. |
| QSPI_QSPCLK_DIV_10 | QSPCLK = PCLK / 10. |
| QSPI_QSPCLK_DIV_11 | QSPCLK = PCLK / 11. |
| QSPI_QSPCLK_DIV_12 | QSPCLK = PCLK / 12. |
| QSPI_QSPCLK_DIV_13 | QSPCLK = PCLK / 13. |
| QSPI_QSPCLK_DIV_14 | QSPCLK = PCLK / 14. |
| QSPI_QSPCLK_DIV_15 | QSPCLK = PCLK / 15. |
| QSPI_QSPCLK_DIV_16 | QSPCLK = PCLK / 16. |
| QSPI_QSPCLK_DIV_17 | QSPCLK = PCLK / 17. |
| QSPI_QSPCLK_DIV_18 | QSPCLK = PCLK / 18. |
| QSPI_QSPCLK_DIV_20 | QSPCLK = PCLK / 20. |
| QSPI_QSPCLK_DIV_22 | QSPCLK = PCLK / 22. |
| QSPI_QSPCLK_DIV_24 | QSPCLK = PCLK / 24. |
| QSPI_QSPCLK_DIV_26 | QSPCLK = PCLK / 26. |
| QSPI_QSPCLK_DIV_28 | |

| | |
|---|---|
| | QSPCLK = PCLK / 28. |
| QSPI_QSPCLK_DIV_30 | QSPCLK = PCLK / 30. |
| QSPI_QSPCLK_DIV_32 | QSPCLK = PCLK / 32. |
| QSPI_QSPCLK_DIV_34 | QSPCLK = PCLK / 34. |
| QSPI_QSPCLK_DIV_36 | QSPCLK = PCLK / 36. |
| QSPI_QSPCLK_DIV_38 | QSPCLK = PCLK / 38. |
| QSPI_QSPCLK_DIV_40 | QSPCLK = PCLK / 40. |
| QSPI_QSPCLK_DIV_42 | QSPCLK = PCLK / 42. |
| QSPI_QSPCLK_DIV_44 | QSPCLK = PCLK / 44. |
| QSPI_QSPCLK_DIV_46 | QSPCLK = PCLK / 46. |
| QSPI_QSPCLK_DIV_48 | QSPCLK = PCLK / 48. |

## Function Documentation

### ◆ R_QSPI_Open()

fsp_err_t R_QSPI_Open ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_cfg_t const *const *p_cfg* )

Open the QSPI driver module. After the driver is open, the QSPI can be accessed like internal flash memory starting at address 0x60000000.

Implements spi_flash_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | The parameter p_instance_ctrl or p_cfg is NULL. |
| FSP_ERR_ALREADY_OPEN | Driver has already been opened with the same p_instance_ctrl. |

### ◆ R_QSPI_DirectWrite()

fsp_err_t R_QSPI_DirectWrite ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write* )

Writes raw data directly to the QSPI.

Implements spi_flash_api_t::directWrite.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_DirectRead()

fsp_err_t R_QSPI_DirectRead ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes* )

Reads raw data directly from the QSPI. This API can only be called after R_QSPI_DirectWrite with read_after_write set to true.

Implements spi_flash_api_t::directRead.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function must be called after R_QSPI_DirectWrite with read_after_write set to true. |

### ◆ R_QSPI_DirectTransfer()

fsp_err_t R_QSPI_DirectTransfer ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction* )

Read/Write raw data directly with the OctaFlash/OctaRAM device. Unsupported by QSPI.

Implements spi_flash_api_t::directTransfer.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by QSPI. |

### ◆ R_QSPI_XipEnter()

fsp_err_t R_QSPI_XipEnter ( spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_XipExit()

fsp_err_t R_QSPI_XipExit ( spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_Write()

fsp_err_t R_QSPI_Write ( spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count* )

Program a page of data to the flash.

Implements spi_flash_api_t::write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The flash was programmed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_Erase()

fsp_err_t R_QSPI_Erase ( spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count* )

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The command to erase the flash was executed successfully. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_device_address is NULL, or byte_count doesn't match an erase size defined in spi_flash_cfg_t, or device is in XIP mode. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |
| FSP_ERR_DEVICE_BUSY | The device is busy. |

### ◆ R_QSPI_StatusGet()

fsp_err_t R_QSPI_StatusGet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status* )

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The write status is in p_status. |
| FSP_ERR_ASSERTION | p_instance_ctrl or p_status is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_MODE | This function can't be called when XIP mode is enabled. |

### ◆ R_QSPI_BankSet()

fsp_err_t R_QSPI_BankSet ( spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank* )

Selects the bank to access. A bank is a 64MB sliding access window into the QSPI device flash memory space. To access chip address 0x4000000, select bank 1, then read from internal flash address 0x60000000. To access chip address 0x8001000, select bank 2, then read from internal flash address 0x60001000.

This function is not required for memory devices less than or equal to 512 Mb (64MB).

Implements spi_flash_api_t::bankSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Bank successfully selected. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

### ◆ R_QSPI_SpiProtocolSet()

fsp_err_t R_QSPI_SpiProtocolSet ( spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol* )

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | SPI protocol updated on MCU peripheral. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid SPI protocol requested. |

### ◆ R_QSPI_AutoCalibrate()

fsp_err_t R_QSPI_AutoCalibrate ( spi_flash_ctrl_t * *p_ctrl*)

Auto-calibrate the OctaRAM device using the preamble pattern. Unsupported by QSPI. Implements spi_flash_api_t::autoCalibrate.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | API not supported by QSPI |

### ◆ R_QSPI_Close()

fsp_err_t R_QSPI_Close ( spi_flash_ctrl_t * *p_ctrl*)

Close the QSPI driver module.

Implements spi_flash_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |
| FSP_ERR_ASSERTION | p_instance_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |

## 5.2.17.18 SD/MMC (r_sdhi)

Modules » Storage

## Functions

| | |
|---|---|
| fsp_err_t | R_SDHI_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg) |
| fsp_err_t | R_SDHI_MediaInit (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_device_t *const p_device) |
| fsp_err_t | R_SDHI_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_ReadIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address) |
| fsp_err_t | R_SDHI_WriteIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write) |
| fsp_err_t | R_SDHI_ReadIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode) |
| fsp_err_t | R_SDHI_WriteIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode) |
| fsp_err_t | R_SDHI_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable) |
| fsp_err_t | R_SDHI_StatusGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_status_t *const p_status) |
| fsp_err_t | R_SDHI_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const start_sector, uint32_t const sector_count) |
| fsp_err_t | R_SDHI_CallbackSet (sdmmc_ctrl_t *const p_api_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_SDHI_Close (sdmmc_ctrl_t *const p_api_ctrl) |

## Detailed Description

Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface.

# Overview

### Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity) and eMMC (embedded Multi Media Card)
    - Supports reading, writing and erasing SD memory devices
    - Supports 1, 4 or 8-bit data bus (8-bit bus is supported for eMMC only)
    - Supports detection of device write protection (SD cards only)
    - Supports high speed mode
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device
- Supports hardware acceleration using DMAC or DTC
- Supports callback notification when an operation completes or an error occurs

# Configuration

### Build Time Configurations for r_sdhi

The following build time configurations are defined in fsp_cfg/r_sdhi_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Unaligned Access Support | • Disabled<br>• Enabled | Enabled | If enabled, code for supporting buffers that are not aligned on a 4-byte boundary is included in the build. Only disable this if all buffers passed to the driver are 4-byte aligned. |
| SD Support | • Disabled<br>• Enabled | Enabled | If selected code for SD card support is included in the build. |
| eMMC Support | • Disabled<br>• Enabled | Disabled | If selected code for eMMC device support is included in the build. |

### Configurations for Storage > SD/MMC (r_sdhi)

This module can be added to the Stacks tab via New Stack > Storage > SD/MMC (r_sdhi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_sdmmc0 | Module name. |

| | | | |
|---|---|---|---|
| Channel | Value must be a non-negative integer | 0 | Select the channel. |
| Bus Width | MCU Specific Options | | Select the bus width. |
| Card Detection | • Not Used<br>• CD Pin | CD Pin | Select the card detection method. |
| Write Protection | • Not Used<br>• WP Pin | WP Pin | Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Access Interrupt Priority | MCU Specific Options | | Select the access interrupt priority. |
| Card Interrupt Priority | MCU Specific Options | | Select the card interrupt priority. |
| DTC Interrupt Priority | MCU Specific Options | | Select the DTC interrupt priority. |

**Interrupt Configurations:**

The following interrupts are required to use the r_sdhi module:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional and only available on MCU packages that have the SDnCD pin (n = channel number).

**Clock Configuration**

The SDMMC MCU peripheral (SDHI) uses the PCLKA for its clock source. The SDMMC driver selects the optimal built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the device obtained at media initialization.

**Pin Configuration**

The SDMMC driver supports the following pins (n = channel number):

- SDnCLK
- SDnCMD
- SDnDAT0
- SDnDAT1
- SDnDAT2
- SDnDAT3
- SDnDAT4 (not available on all MCUs)
- SDnDAT5 (not available on all MCUs)
- SDnDAT6 (not available on all MCUs)
- SDnDAT7 (not available on all MCUs)
- SDnCD (not available on all MCUs)
- SDnWP

The drive capacity for each pin should be set to "Medium" or "High" for most hardware designs. This can be configured in the **Pins** tab of the RA Configuration editor by selecting the pin under Pin Selection -> Ports.

# Usage Notes

### Card Detection

When Card Detection is configured to "CD Pin" in the RA Configuration editor, interrupt flags are cleared and card detection is enabled during R_SDHI_Open().

R_SDHI_StatusGet() can be called to retrieve the current status of the card (including whether a card is present). If the Card Interrupt Priority is enabled, a callback is called when a card is inserted or removed.

If a card is removed and reinserted, R_SDHI_MediaInit() must be called before reading from the card or writing to the card.

*Note*

*R_SDHI_StatusGet() should be used to initially determine the card state after opening the interface.*

### DMA Request Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the DMA Request interrupt. This blocks all other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

### Timing Notes for R_SDHI_MediaInit

The R_SDHI_MediaInit() API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1 bit and a bus speed of 400 kHz or less.

### Limitations

Developers should be aware of the following limitations when using the SDHI:

### Blocking Calls

The following functions block execution until the response is received for at least one command:

- R_SDHI_MediaInit
- R_SDHI_Erase

Once the function returns the status of the operation can be determined via R_SDHI_StatusGet or through receipt of a callback.

*Note*

*Due to the variability in clocking configurations it is recommended to determine blocking delays experimentally on the target system.*

### Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdhi driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred and a software copy is used to move data to the destination buffer.

## Examples

### Basic Example

This is a basic example of minimal use of the r_sdhi in an application.

```
uint8_t g_dest[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void r_sdhi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < SDHI_MAX_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the SDHI driver. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
    assert(FSP_SUCCESS == err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
```

```
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
    err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
    err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
}
/* The callback is called when a transfer completes. */
void r_sdhi_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = 1;
    }
}
```

### Card Detection Example

This is an example of using SDHI when the card may not be plugged in. The card detection interrupt must be enabled to use this example.

```
bool g_card_inserted = false;
void r_sdhi_card_detect_example (void)
{
```

```
/* Open the SDHI driver. This enables the card detection interrupt. */
fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Check if card is inserted. */
sdmmc_status_t status;
   err = R_SDHI_StatusGet(&g_sdmmc0_ctrl, &status);
   assert(FSP_SUCCESS == err);
if (!status.card_inserted)
   {
while (!g_card_inserted)
     {
/* Wait for a card insertion interrupt. */
     }
   }
/* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
/* Initialize the SD card after card insertion is detected. */
   err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
   assert(FSP_SUCCESS == err);
}
/* The callback is called when a card detection event occurs if the card detection
interrupt is enabled. */
void r_sdhi_card_detect_example_callback (sdmmc_callback_args_t * p_args)
{
if (SDMMC_EVENT_CARD_INSERTED == p_args->event)
   {
      g_card_inserted = true;
   }
if (SDMMC_EVENT_CARD_REMOVED == p_args->event)
   {
```

```
        g_card_inserted = false;
    }
}
```

### Function Documentation

#### ◆ R_SDHI_Open()

fsp_err_t R_SDHI_Open ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_cfg_t const *const *p_cfg* )

Opens the driver. Resets SDHI, and enables card detection interrupts if card detection is enabled. R_SDHI_MediaInit must be called after this function before any other functions can be used.

Implements sdmmc_api_t::open().

Example:

```
/* Open the SDHI driver. */
fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is now open. |
| FSP_ERR_ASSERTION | Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. |
| FSP_ERR_ALREADY_OPEN | Driver has already been opened with this instance of the control structure. |
| FSP_ERR_IRQ_BSP_DISABLED | Access interrupt is not enabled. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel does not exist on this MCU. |

### ◆ R_SDHI_MediaInit()

fsp_err_t R_SDHI_MediaInit ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_device_t *const *p_device* )

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements sdmmc_api_t::mediaInit().

Example:

```
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Module is now ready for read/write access. |
| FSP_ERR_ASSERTION | Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_INIT_FAILED | Device was not identified as an SD card or eMMC device. |
| FSP_ERR_RESPONSE | Device responded with an error. |
| FSP_ERR_TIMEOUT | Device did not respond. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_Read()

fsp_err_t R_SDHI_Read ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements sdmmc_api_t::read().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Example:

```
err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Data read successfully. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |

### ◆ R_SDHI_Write()

fsp_err_t R_SDHI_Write ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements sdmmc_api_t::write().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written and the device is no longer holding DAT0 low to indicate it is busy.

Example:

```
err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Card write finished successfully. |
| FSP_ERR_ASSERTION | Handle or Source address is NULL. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_DEVICE_BUSY | Driver is busy with a previous operation. |
| FSP_ERR_CARD_WRITE_PROTECTED | SD card is Write Protected. |
| FSP_ERR_WRITE_FAILED | Write operation failed. |

### ◆ R_SDHI_ReadIo()

fsp_err_t R_SDHI_ReadIo ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address* )

The Read function reads a one byte register from an SDIO card. Implements sdmmc_api_t::readIo().

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | SDIO is not supported. |

### ◆ R_SDHI_WriteIo()

fsp_err_t R_SDHI_WriteIo ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*, sdmmc_io_write_mode_t const *read_after_write* )

Writes a one byte register to an SDIO card. Implements sdmmc_api_t::writeIo().

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | SDIO is not supported. |

### ◆ R_SDHI_ReadIoExt()

fsp_err_t R_SDHI_ReadIoExt ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *function*, uint32_t const *address*, uint32_t *const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode* )

Reads data from an SDIO card function. Implements sdmmc_api_t::readIoExt().

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | SDIO is not supported. |

### ◆ R_SDHI_WriteIoExt()

fsp_err_t R_SDHI_WriteIoExt ( sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *function*, uint32_t const *address*, uint32_t const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode* )

Writes data to an SDIO card function. Implements sdmmc_api_t::writeIoExt().

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | SDIO is not supported. |

### ◆ R_SDHI_IoIntEnable()

fsp_err_t R_SDHI_IoIntEnable ( sdmmc_ctrl_t *const *p_api_ctrl*, bool *enable* )

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

***Note***

*This function is not supported.*

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | SDIO is not supported. |

### ◆ R_SDHI_StatusGet()

fsp_err_t R_SDHI_StatusGet ( sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_status_t *const *p_status* )

Provides driver status. Implements sdmmc_api_t::statusGet().

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Status stored in p_status. |
| FSP_ERR_ASSERTION | NULL pointer. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |

### ◆ R_SDHI_Erase()

fsp_err_t R_SDHI_Erase ( sdmmc_ctrl_t *const *p_api_ctrl*, uint32_t const *start_sector*, uint32_t const *sector_count* )

Erases sectors of an SD card or eMMC device. Implements sdmmc_api_t::erase().

This function blocks until the erase command is sent. Poll the status to determine when erase is complete.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Erase operation requested. |
| FSP_ERR_ASSERTION | A required pointer is NULL or an argument is invalid. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |
| FSP_ERR_CARD_NOT_INITIALIZED | Card was unplugged. |
| FSP_ERR_CARD_WRITE_PROTECTED | SD card is Write Protected. |
| FSP_ERR_RESPONSE | Device responded with an error. |
| FSP_ERR_TIMEOUT | Device did not respond. |
| FSP_ERR_DEVICE_BUSY | Device is holding DAT0 low (device is busy) or another operation is ongoing. |

### ◆ R_SDHI_CallbackSet()

fsp_err_t R_SDHI_CallbackSet ( sdmmc_ctrl_t *const *p_api_ctrl*, void(*)(sdmmc_callback_args_t *) *p_callback*, void *const *p_context*, sdmmc_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements sdmmc_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_SDHI_Close()

| fsp_err_t R_SDHI_Close ( sdmmc_ctrl_t *const *p_api_ctrl*) |
|---|
| Closes an open SD/MMC device. Implements sdmmc_api_t::close(). |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | The parameter p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | Driver has not been initialized. |

## 5.2.17.19 Virtual EEPROM on Flash (rm_vee_flash)

Modules » Storage

### Functions

| | |
|---|---|
| fsp_err_t | RM_VEE_FLASH_Open (rm_vee_ctrl_t *const p_api_ctrl, rm_vee_cfg_t const *const p_cfg) |
| fsp_err_t | RM_VEE_FLASH_RecordWrite (rm_vee_ctrl_t *const p_api_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t const num_bytes) |
| fsp_err_t | RM_VEE_FLASH_RefDataWrite (rm_vee_ctrl_t *const p_api_ctrl, uint8_t const *const p_ref_data) |
| fsp_err_t | RM_VEE_FLASH_RecordPtrGet (rm_vee_ctrl_t *const p_api_ctrl, uint32_t const rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes) |
| fsp_err_t | RM_VEE_FLASH_RefDataPtrGet (rm_vee_ctrl_t *const p_api_ctrl, uint8_t **const pp_ref_data) |
| fsp_err_t | RM_VEE_FLASH_Refresh (rm_vee_ctrl_t *const p_api_ctrl) |
| fsp_err_t | RM_VEE_FLASH_Format (rm_vee_ctrl_t *const p_api_ctrl, uint8_t const *const p_ref_data) |
| fsp_err_t | RM_VEE_FLASH_CallbackSet (rm_vee_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void *const p_context, rm_vee_callback_args_t *const p_callback_memory) |
| fsp_err_t | RM_VEE_FLASH_StatusGet (rm_vee_ctrl_t *const p_api_ctrl, rm_vee_status_t *const p_status) |

fsp_err_t RM_VEE_FLASH_Close (rm_vee_ctrl_t *const p_api_ctrl)

### Detailed Description

Virtual EEPROM on RA MCUs. This module implements the Virtual EEPROM Interface.

# Overview

This VEE module emulates basic EEPROM capabilities. Support is provided for reading and writing both common records and reference data (originally programmed during product assembly or test). A count of the number of segments erased throughout the lifetime of the application is maintained and can be accessed at any time. Wear leveling is handled automatically by the driver.

### Features

- Writing and reading user defined records of any length to data flash.
- Wear leveling is handled automatically.
- Reference data such as calibration data programmed at assembly or test time is preserved.
- Reference data can be updated at run time.
- Fault resilient design.

### Data Flash Segmentation

Wear leveling is handled by changing the location in the data flash where a record is stored every time that it is updated. This change in physical location of the record is transparent to the user. Any time an update for a specific record ID is written, it is written to the next unused location in data flash and its location is stored in RAM for quick look-up later. When required, only the most recent version of these records is automatically copied to the next blank segment in data flash. The data flash area is divided into a number of equal-size segments. There is only one segment active at a time. A segment contains two areas- the record area (which is the vast majority of the segment) and the reference data area which contains optional data typically programmed during assembly or final test. Records and updated reference data are written to this segment until one of the two areas becomes full. The record area must be able to hold at least one of every record ID possible and still have space left over for record updates.

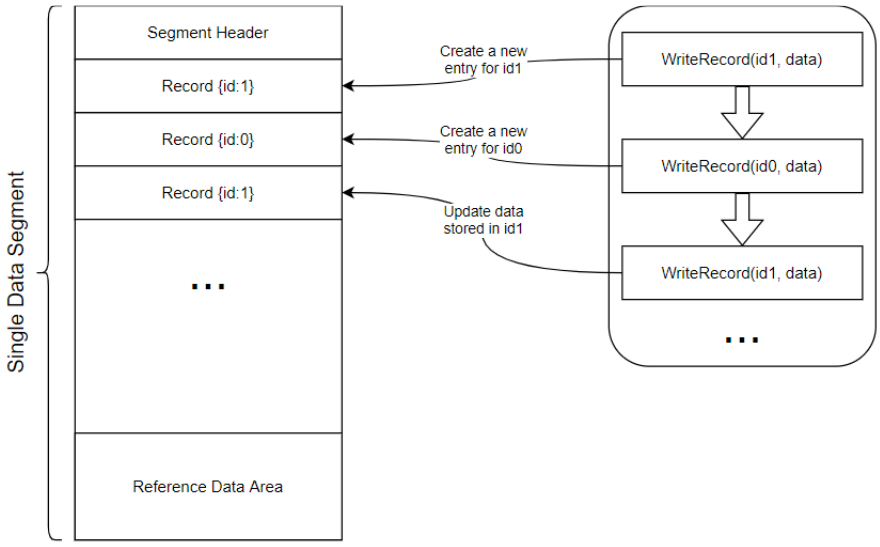



Figure 330: Segment Data Format

When a segment does not have sufficient space for additional records or updated reference data, a Refresh occurs. This process copies the most recent record for each ID as well as the latest version of reference data (if any) to the next segment. The very first time VEE runs on an MCU, it marks the last segment as active whether there is reference data configured or not. The end of VEE data flash area is used to provide an easily identified physical flash address that can be used while programming reference data without requiring Virtual EEPROM middleware.

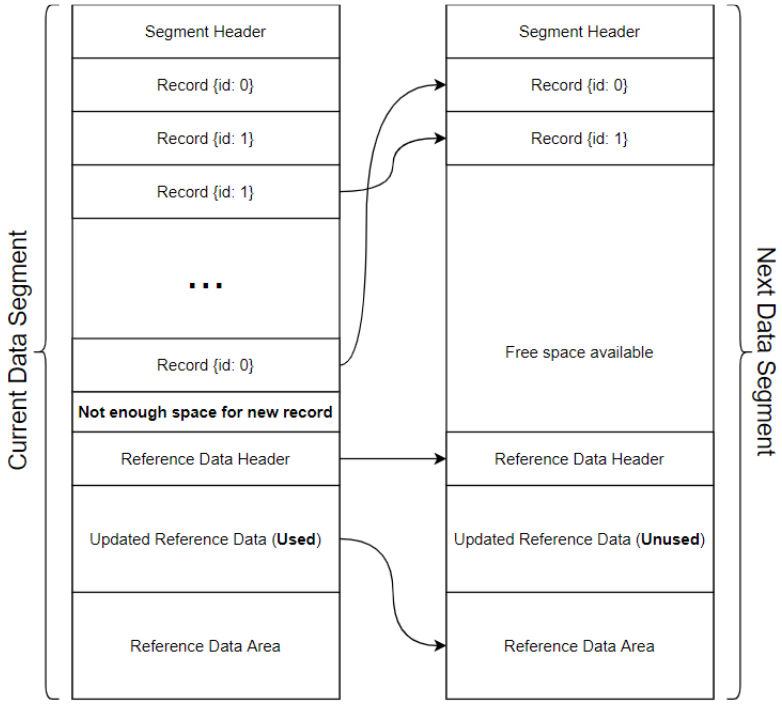



Figure 331: Refresh Operation

### Record Format

Each record begins with a header that contains the record size, followed by the data, and the trailer.

The trailer contains a validation code which is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the record data passed to the driver. Padding is added between the end of user data and the trailer to ensure the trailer is aligned properly.

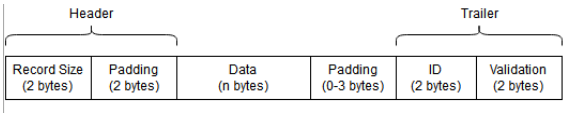



Figure 332: Record Format

### Reference Data Area

VEE can be configured for the presence of reference data. The original programmed reference data must be located at the end of the VEE data flash area. An area of equal size is reserved below this in case updated reference data becomes available later. Below that is a header which indicates whether the update area has been written to.

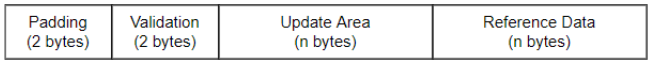



Figure 333: Reference Data Area Format

Just as with records, the validation code is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the updated reference data passed to the driver.

### Fault Tolerance

The Virtual EEPROM has a fault tolerant design. If for any reason an operation fails before it is completed the next time the module is opened a refresh will occur. Any corrupted data will be discarded.

# Configuration

### Build Time Configurations for rm_vee_flash

The following build time configurations are defined in fsp_cfg/rm_vee_flash_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Reference Data Support | • Enabled<br>• Disabled | Disabled | Support writing reference data to the end of the segment. |
| Refresh Buffer Size | Value must be an integer greater than 0 and a multiple of 4 bytes. | 32 | The size of the internal buffer used to copying data from one flash segment to another |

| | | | |
|---|---|---|---|
| | | | during a refresh operation. This is required because data flash to data flash transfers are not supported by the hardware. |

### Configurations for Storage > Virtual EEPROM on Flash (rm_vee_flash)

This module can be added to the Stacks tab via New Stack > Storage > Virtual EEPROM on Flash (rm_vee_flash). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_vee0 | Module name. |
| Record Max ID | Value must be an integer. | 16 | Set this value to the highest record ID in use. |
| Number of Segments | Value must be an integer. | 2 | Set value to number of segments desired in data flash (minimum 2). The fewer the segments, the fewer refreshes occur, but the longer refreshes take to complete (erase time). |
| Start Address | Manual Entry | BSP_FEATURE_FLASH_DATA_FLASH_START | Start address of the flash area used by Virtual EEPROM. |
| Total Size | Manual Entry | BSP_DATA_FLASH_SIZE_BYTES | The total size (In bytes) of the flash area used by Virtual EEPROM. |
| Reference Data Size | Value must be an integer. | 0 | The size of the reference area (In bytes) used by Virtual EEPROM. |
| Callback | Name must be a valid C symbol | vee_callback | A user callback function can be provided. If this callback function is provided, it will be called from the flash interrupt service routine (ISR). |

### Clock Configuration

There is no clock configuration for the RM_VEE_FLASH module.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

A refresh buffer is required to copy data between segments. Data flash cannot be simultaneously read from and written to. Data will be temporarily copied into RAM during refresh operations.

## Examples

### Basic Example

This is a basic example of minimal use of the RM_VEE_FLASH module in an application.

```
volatile bool callback_called = false;
/* Record ID to use for storing pressure data. */
#define ID_PRESSURE (0U)
/* Example data structure. */
typedef struct st_pressure
{
    uint32_t timestamp;
    uint16_t low;
    uint16_t average;
    uint16_t high;
} pressure_t;
void rm_vee_example ()
{
 /* Open the Virtual EEPROM Module. */
 fsp_err_t err = RM_VEE_FLASH_Open(&g_vee_ctrl, &g_vee_cfg);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Read pressure data from external sensor. */
    pressure_t pressure_data;
    get_pressure_data(&pressure_data);
 /* Write the pressure data to a Virtual EEPROM Record. */
```

```
    err = RM_VEE_FLASH_RecordWrite(&g_vee_ctrl, ID_PRESSURE, (uint8_t *)
&pressure_data, sizeof(pressure_t));
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Wait for the Virtual EEPROM callback to indicate it finished writing data. */
 while (false == callback_called)
    {
       ;
    }
 /* Get a pointer to the record that is stored in data flash. */
    uint32_t    length;
    pressure_t * p_pressure_data;
    err = RM_VEE_FLASH_RecordPtrGet(&g_vee_ctrl, ID_PRESSURE, (uint8_t **)
&p_pressure_data, &length);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Close the Virtual EEPROM Module. */
    err = RM_VEE_FLASH_Close(&g_vee_ctrl);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
}
void rm_vee_tests_callback (rm_vee_callback_args_t * p_args)
{
    callback_called = true;
 FSP_PARAMETER_NOT_USED(p_args);
}
```

## Data Structures

| | |
|---|---|
| struct | rm_vee_flash_cfg_t |

struct rm_vee_flash_instance_ctrl_t

## Data Structure Documentation

### ◆ rm_vee_flash_cfg_t

| struct rm_vee_flash_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| flash_instance_t const * | p_flash | Pointer to a flash instance. |

### ◆ rm_vee_flash_instance_ctrl_t

| struct rm_vee_flash_instance_ctrl_t |
|---|
| Instance control block. This is private to the FSP and should not be used or modified by the application. |

## Function Documentation

### ◆ RM_VEE_FLASH_Open()

fsp_err_t RM_VEE_FLASH_Open ( rm_vee_ctrl_t *const *p_api_ctrl*, rm_vee_cfg_t const *const *p_cfg* )

Open the RM_VEE_FLASH driver module

Implements rm_vee_api_t::open

Initializes the driver's internal structures and opens the Flash driver. The Flash driver must be closed prior to opening VEE. The error code FSP_SUCCESS_RECOVERY indicates that VEE detected corrupted data; most likely due to a power loss during a data flash write or erase. In these cases, an automatic internal Refresh is performed and the partially written data is lost.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. FSP_SUCCESS_RECOVERY changed to FSP_SUCCESS |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_ALREADY_OPEN | This function has already been called. |
| FSP_ERR_PE_FAILURE | This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware. |
| FSP_ERR_TIMEOUT | Interrupts disabled outside of VEE |
| FSP_ERR_NOT_INITIALIZED | Corruption found. A refresh is required. |
| FSP_ERR_INVALID_ARGUMENT | The supplied configuration is invalid. |

### ◆ RM_VEE_FLASH_RecordWrite()

fsp_err_t RM_VEE_FLASH_RecordWrite ( rm_vee_ctrl_t *const *p_api_ctrl*, uint32_t const *rec_id*, uint8_t const *const *p_rec_data*, uint32_t const *num_bytes* )

Writes a record to data flash.

Implements rm_vee_api_t::recordWrite

This function writes num_bytes of data pointed to by p_rec_data to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes. This includes exiting the calling function when the data buffer is a local variable (stack may be used by another function and corrupt the data buffer contents).

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write started successfully. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_INVALID_ARGUMENT | An argument contains an illegal value. |
| FSP_ERR_INVALID_MODE | The operation cannot be started in the current mode. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_PE_FAILURE | This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware. |
| FSP_ERR_TIMEOUT | Flash write timed out (Should not be possible when flash bgo is used). |
| FSP_ERR_NOT_INITIALIZED | Corruption found. A refresh is required. |

### ◆ RM_VEE_FLASH_RefDataWrite()

fsp_err_t RM_VEE_FLASH_RefDataWrite ( rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_ref_data* )

Writes new Reference data to the reference update area.

Implements rm_vee_api_t::refDataWrite

This function writes VEE_CFG_REF_DATA_SIZE bytes pointed to by p_ref_data to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Write started successfully. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_INVALID_MODE | The operation cannot be started in the current mode. |
| FSP_ERR_PE_FAILURE | This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware. |
| FSP_ERR_TIMEOUT | Flash write timed out (Should not be possible when flash bgo is used). |
| FSP_ERR_UNSUPPORTED | Reference data is not supported in the current configuration. |
| FSP_ERR_NOT_INITIALIZED | Corruption found. A refresh is required. |

### ◆ RM_VEE_FLASH_RecordPtrGet()

fsp_err_t RM_VEE_FLASH_RecordPtrGet ( rm_vee_ctrl_t *const *p_api_ctrl*, uint32_t const *rec_id*, uint8_t **const *pp_rec_data*, uint32_t *const *p_num_bytes* )

Gets a pointer to the most recent record data.

Implements rm_vee_api_t::recordPtrGet

This function sets the argument pointer to the most recent version of the record data in flash. Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at p_ref_data must be completed prior to initiating any type of Flash write.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_ASSERTION | p_ref_data is NULL. |
| FSP_ERR_INVALID_ARGUMENT | Record data not configured. |
| FSP_ERR_NOT_FOUND | The record associated with the requested ID could not be found. |

### ◆ RM_VEE_FLASH_RefDataPtrGet()

fsp_err_t RM_VEE_FLASH_RefDataPtrGet ( rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t **const *pp_ref_data* )

Gets a pointer to the most recent reference data.

Implements rm_vee_api_t::recordPtrGet

This function sets the argument pointer to the most recent version of the reference data in flash. Flash cannot be accessed for reading and writing at the same time.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_UNSUPPORTED | Reference data is not supported in the current configuration. |
| FSP_ERR_NOT_FOUND | No reference data was found. |

### ◆ RM_VEE_FLASH_Refresh()

fsp_err_t RM_VEE_FLASH_Refresh ( rm_vee_ctrl_t *const *p_api_ctrl*)

Manually start a refresh operation

Implements rm_vee_api_t::refresh

This function is used to start a segment Refresh at any time. The Refresh process by default occurs automatically when no more record or reference data space is available and a Write is requested. However, the app may desire to force a refresh when it knows it is running low on space and large amounts of data are about to be recorded.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_PE_FAILURE | This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware. |
| FSP_ERR_INVALID_MODE | The operation cannot be started in the current mode. |
| FSP_ERR_TIMEOUT | Flash write timed out (Should not be possible when flash bgo is used). |
| FSP_ERR_NOT_INITIALIZED | Corruption found. A refresh is required. |

### ◆ RM_VEE_FLASH_Format()

fsp_err_t RM_VEE_FLASH_Format ( rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_ref_data* )

Start a manual format operation.

Implements rm_vee_api_t::format

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_IN_USE | Last API call still executing. |
| FSP_ERR_PE_FAILURE | This error indicates that a flash programming, erase, or blankcheck operation has failed |
| FSP_ERR_ASSERTION | An input parameter is NULL. |
| FSP_ERR_TIMEOUT | Flash write timed out (Should not be possible when flash bgo is used). |
| FSP_ERR_NOT_INITIALIZED | Corruption found. A refresh is required. |

### ◆ RM_VEE_FLASH_CallbackSet()

fsp_err_t RM_VEE_FLASH_CallbackSet ( rm_vee_ctrl_t *const *p_api_ctrl*, void(*)(rm_vee_callback_args_t *) *p_callback*, void *const *p_context*, rm_vee_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure.

Implements rm_vee_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ RM_VEE_FLASH_StatusGet()

fsp_err_t RM_VEE_FLASH_StatusGet ( rm_vee_ctrl_t *const *p_api_ctrl*, rm_vee_status_t *const *p_status* )

Get the current status of the driver.

Implements rm_vee_api_t::statusGet

This command is typically used to verify that the last Write or Refresh command has completed before attempting to perform another API call.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |

### ◆ RM_VEE_FLASH_Close()

fsp_err_t RM_VEE_FLASH_Close ( rm_vee_ctrl_t *const *p_api_ctrl*)

Closes the Flash driver and VEE driver.

Implements rm_vee_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful. |
| FSP_ERR_NOT_OPEN | The module has not been opened. |
| FSP_ERR_ASSERTION | An input parameter is NULL. |

## 5.2.18 System

Modules

### Detailed Description

System Modules.

### Modules

Clock Generation Circuit (r_cgc)

Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface.

| | |
|---|---|
| | Event Link Controller (r_elc)<br>Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface. |
| | I/O Port (r_ioport)<br>Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface. |
| | Inter-Processor Communication (r_ipc)<br>Driver for the IPC peripheral on RA MCUs. This module implements the IPC Interface. |

### 5.2.18.1 Clock Generation Circuit (r_cgc)

Modules » System

**Functions**

| | |
|---|---|
| fsp_err_t | R_CGC_Open (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg) |
| fsp_err_t | R_CGC_ClocksCfg (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg) |
| fsp_err_t | R_CGC_ClockStart (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg) |
| fsp_err_t | R_CGC_ClockStop (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| fsp_err_t | R_CGC_SystemClockSet (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg) |
| fsp_err_t | R_CGC_SystemClockGet (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg) |
| fsp_err_t | R_CGC_ClockCheck (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| fsp_err_t | R_CGC_OscStopDetectEnable (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_OscStopDetectDisable (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_OscStopStatusClear (cgc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_CGC_CallbackSet (cgc_ctrl_t *const p_api_ctrl, |

| | |
|---|---|
| | void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_CGC_Close (cgc_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface.

*Note*

***This module is not required for the initial clock configuration. Initial clock settings are configurable on the **Clocks** tab of the RA Configuration editor. The initial clock settings are applied by the BSP during the startup process before main.***

# Overview

### Features

The CGC module supports runtime modifications of clock settings. Key features include the following:

- Supports changing the system clock source to any of the following options (provided they are supported on the MCU):
  - High-speed on-chip oscillator (HOCO)
  - Middle-speed on-chip oscillator (MOCO)
  - Low-speed on-chip oscillator (LOCO)
  - Main oscillator (external resonator or external clock input frequency)
  - Sub-clock oscillator (external resonator)
  - PLL/PLL2 (not available on all MCUs)
- When the system core clock frequency changes, the following things are updated:
  - The CMSIS standard global variable SystemCoreClock is updated to reflect the new clock frequency.
  - Wait states for ROM and RAM are adjusted to the minimum supported value for the new clock frequency.
  - The operating power control mode is updated to the minimum supported value for the new clock settings.
- Supports starting or stopping any of the system clock sources
- Supports changing dividers for the internal clocks
- Supports the oscillation stop detection feature

### Internal Clocks

The RA microcontrollers have up to seven internal clocks. Not all internal clocks exist on all MCUs. Each clock domain has its own divider that can be updated in R_CGC_SystemClockSet(). The dividers are subject to constraints described in the footnote of the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual.

The internal clocks include:

- System clock (ICLK): core clock used for CPU, flash, internal SRAM, DTC, and DMAC
- PCLKA/PCLKB/PCLKC/PCLKD: Peripheral clocks, refer to the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual to see which peripherals are controlled by which clocks.
- FCLK: Clock source for reading data flash and for programming/erasure of both code and

data flash.
- BCLK: External bus clock

# Configuration

*Note*

*The initial clock settings are configurable on the **Clocks** tab of the RA Configuration editor.*
*There is a configuration to enable the HOCO on reset in the OFS1 settings on the BSP tab.*
*The following clock related settings are configurable in the RA Common section on the BSP tab:*
- *Main Oscillator Wait Time*
- *Main Oscillator Clock Source (external oscillator or crystal/resonator)*
- *Subclock Populated*
- *Subclock Drive*
- *Subclock Stabilization Time (ms)*

*The default stabilization times are determined based on development boards provided by Renesas, but are generally valid for most designs. Depending on the target board hardware configuration and requirements these values may need to be adjusted for reliability or startup speed.*

### Build Time Configurations for r_cgc

The following build time configurations are defined in fsp_cfg/r_cgc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for System > Clock Generation Circuit (r_cgc)

This module can be added to the Stacks tab via New Stack > System > Clock Generation Circuit (r_cgc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Sub Clock Oscillation Stop Detection Settings | | | |
| Enable sub clock oscillator stop detection via the SOSC_STOP interrupt | • Enabled<br>• Disabled | Disabled | Oscillation Stop Detection for Sub Clock enabled. |
| Sub Clock Oscillation Stop Detection Time | Manual Entry | 0 | These bits specify the oscillation stop detection time. It is detected that oscillation has stopped when oscillation has been stopped for (A-2) to (A+1) clock cycles, where A refers to the time specified by these bits. Oscillation stop detection time = Low- |

| | | | |
|---|---|---|---|
| | | | speed on-chip oscillator clock (LOCO) cycle x ((value of OSDCCMP) + 1). |
| Sub Clock Oscillation Stop Detection Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the Sub Clock Oscillation Stop Detection interrupt. |
| Main Oscillation Stop Detection Settings | | | |
| Enable main clock oscillator stop detection via the MOSC_STOP interrupt | • Enabled<br>• Disabled | Disabled | Oscillation Stop Detection for Main Clock enabled. |
| Main Oscillation Stop Detection Time | Manual Entry | 0 | These bits specify the oscillation stop detection time. It is detected that oscillation has stopped when oscillation has been stopped for (A - 2) to (A + 1) clock cycles, where A refers to the time specified by these bits. Oscillation stop detection time = High speed on-chip oscillator clock (HOCO) cycle x ((value of OSDCCMP) + 1). |
| Main Oscillation Stop Detection Interrupt Priority | MCU Specific Options | | [Optional] Select the interrupt priority for the Main Oscillation Stop Detection interrupt. |
| Oscillator Stop Detection Settings | | | |
| Clock Switch Enable for Oscillation Stop Detected of SDADCCLK | • Enabled<br>• Disabled | Disabled | The OSTDCSE bit specifies the source clock of the 24-bit Sigma-delta A/D Converter Clock switched to HOCO when oscillation stop detected. |
| Name | Name must be a valid C symbol | g_cgc0 | Module name. |
| Enable main oscillator stop detection via the NMI interrupt | • Enabled<br>• Disabled | Enabled | Oscillation Stop Detection for System Clock enabled. |

| | | | |
|---|---|---|---|
| Oscillation Stop Detection Callback | Name must be a valid C symbol | NULL | A user callback function must be provided if oscillation stop detection for System Clock/Main Oscillation/Sub Clock Oscillation is used. If this callback function is provided, it is called from the NMI handler if the main oscillator stops. |

### Clock Configuration

This module is used to configure the system clocks. There are no module specific clock configurations required to use it.

### Pin Configuration

The CGC module controls the output of the CLOCKOUT signal.

If an external oscillator is used the XTAL and EXTAL pins must be configured accordingly. When running from an on chip oscillator there is no requirement for the main clock external oscillator. In this case, the XTAL and EXTAL pins can be set to a different function in the RA Configuration editor.

The functionality of the subclock external oscillator pins XCIN and XCOUT is fixed.

## Usage Notes

### Oscillation Stop Detection

CGC driver supports following oscillation stop detection function.

### NMI Interrupt

The CGC driver uses the NMI for oscillation stop detection of the main oscillator after R_CGC_OscStopDetectEnable is called. The NMI is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during R_CGC_Open() is called.

### Main Oscillator Stop Detection Interrupt

A main clock oscillation stop detection interrupt (MOSTD_STOP) is generated when the Oscillation Stop Detection for Main Clock is enabled. The main oscillation stop detection interrupt is a maskable interrupt. When the MOSTD_STOP is triggered, the callback function registered during R_CGC_Open() is called.

### Subclk Oscillator Stop Detection Interrupt

A sub clock oscillation stop detection interrupt (SOSC_STOP) is generated when the Oscillation Stop Detection for Sub Clock is enabled. The sub clock oscillation stop detection interrupt is a maskable interrupt. When the SOSC_STOP is triggered, the callback function registered during R_CGC_Open() is called.

### ADC Clock Switching

When Clock Switch Enable for Oscillation Stop Detected is set.

- When MOSC stop is detected when MOSC is selected for SDADCCLK, SDADCCLK is switched to HOCO.
- When SOSC stop is detected when PLL is selected for SDADCCLK, SDADCCLK is switched to HOCO.

### Starting or Stopping the Subclock

If the Subclock Populated property is set to Populated on the BSP configuration tab, then the subclock is started in the BSP startup routine. Otherwise, it is stopped in the BSP startup routine. Starting and stopping the subclock at runtime is not recommended since the stabilization requirements typically negate the negligible power savings.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take up to several seconds to stabilize. RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. In this case the default wait time is 1000ms (1 second). When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation. Because there is no hardware stabilization status bit for the subclock R_CGC_ClockCheck cannot be used to optimize this wait.

Changing the subclock state during R_CGC_ClocksCfg() is not supported.

### Low Power Operation

If "Use Low Voltage Mode" is enabled in the BSP MCU specific properties (not available on all MCUs), the MCU is always in low voltage mode and no other power modes are considered. The following conditions must be met for the MCU to run in low voltage mode:

- Requires HOCO to be running, so HOCO cannot be stopped in low voltage mode
- Requires PLL to be stopped, so PLL APIs are not available in low voltage mode
- Requires ICLK <= 4 MHz
- If oscillation stop detection is used, dividers of 1 or 2 cannot be used for any clock

If "Use Low Voltage Mode" is not enabled, the MCU applies the lowest power mode by searching through the following list in order and applying the first power mode that is supported under the current conditions:

- Subosc-speed mode (lowest power)
  - Requires system clock to be LOCO or subclock
  - Requires MOCO, HOCO, main oscillator, and PLL (if present) to be stopped
  - Requires ICLK and FCLK dividers to be 1
- Low-speed mode
  - Requires PLL to be stopped
  - Requires ICLK <= 1 MHz
  - If oscillation stop detection is used, dividers of 1, 2, 4, or 8 cannot be used for any clock
- Middle-speed mode (not supported on all MCUs)
  - Requires ICLK <= 8 MHz
- High-speed mode
  - Default mode if no other operating mode is supported

Refer to the section "Function for Lower Operating Power Consumption" in the "Low Power Modes" chapter of the hardware manual for MCU specific information about operating power control modes.

*Note*

*The DCDC regulator (if present) is only available in Middle- and High-speed modes. The BSP will automatically switch between DCDC and LDO when switching between compatible and incompatible modes if the DCDC regulator is in use. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).*

When low voltage mode is not used, the following functions adjust the operating power control mode to ensure it remains within the hardware specification and to ensure the MCU is running at the optimal operating power control mode:

- R_CGC_ClockStart()
- R_CGC_ClockStop()
- R_CGC_SystemClockSet()
- R_CGC_OscStopDetectEnable()
- R_CGC_OscStopDetectDisable()

*Note*

*FSP APIs, including these APIs, are not thread safe. These APIs and any other user code that modifies the operating power control mode must not be allowed to interrupt each other. Proper care must be taken during application design if these APIs are used in threads or interrupts to ensure this constraint is met.*

No action is required by the user of these APIs. This section is provided for informational purposes only.

# Examples

### Basic Example

This is a basic example of minimal use of the CGC in an application.

```
void cgc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to LOCO for power saving. */
 /* Start the LOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
    assert(FSP_SUCCESS == err);
 /* Wait for the LOCO stabilization wait time.
  *
```

```
  * NOTE: The MOCO, LOCO and subclock do not have stabilization status bits, so any
stabilization time must be
  * performed via a software wait when starting these oscillators. For all other
oscillators, R_CGC_ClockCheck can
  * be used to verify stabilization status.
  */
 R_BSP_SoftwareDelay(BSP_FEATURE_CGC_LOCO_STABILIZATION_MAX_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_1,
#else
 #if BSP_FEATURE_CGC_HAS_MOCO
       .moco_divider         = CGC_SYS_CLOCK_DIV_2,
 #endif
       .hoco_divider         = CGC_SYS_CLOCK_DIV_1,
 #if BSP_FEATURE_CGC_HAS_MOSC
       .mosc_divider         = CGC_SYS_CLOCK_DIV_4,
 #endif
```

```
#endif
    };
 /* Switch the system clock to LOCO. */
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    assert(FSP_SUCCESS == err);
}
```

### Configuring Multiple Clocks

This example demonstrates switching to a new source clock and stopping the previous source clock in a single function call using R_CGC_ClocksCfg().

```
void cgc_clocks_cfg_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to PLL running from the main oscillator. */
 /* Assuming the system clock is MOCO, switch to HOCO. */
 cgc_clocks_cfg_t clocks_cfg;
#if !BSP_FEATURE_CGC_REGISTER_SET_B
    clocks_cfg.system_clock                    = CGC_CLOCK_PLL;
    clocks_cfg.pll_state                       = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock            = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier              = CGC_PLL_MUL_10_0;   // unused
    clocks_cfg.pll_cfg.divider                 = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
```

```
#else
    clocks_cfg.system_clock = CGC_CLOCK_HOCO;
 #if BSP_FEATURE_CGC_HAS_MOCO
    clocks_cfg.divider_cfg.moco_divider = CGC_SYS_CLOCK_DIV_4;
 #endif
    clocks_cfg.divider_cfg.hoco_divider = CGC_SYS_CLOCK_DIV_2;
 #if BSP_FEATURE_CGC_HAS_MOSC
    clocks_cfg.divider_cfg.mosc_divider = CGC_SYS_CLOCK_DIV_1;
 #endif
#endif
    clocks_cfg.mainosc_state = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state    = CGC_CLOCK_CHANGE_START;
#if BSP_FEATURE_CGC_HAS_MOCO
    clocks_cfg.moco_state = CGC_CLOCK_CHANGE_STOP;
#else
    clocks_cfg.moco_state = CGC_CLOCK_CHANGE_NONE;
#endif
    clocks_cfg.loco_state = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* Assuming the system clock is HOCO, switch to PLL running from main oscillator and
stop MOCO. */
    clocks_cfg.system_clock                    = CGC_CLOCK_PLL;
    clocks_cfg.pll_state                       = CGC_CLOCK_CHANGE_START;
    clocks_cfg.pll_cfg.source_clock            = CGC_CLOCK_MAIN_OSC;
    clocks_cfg.pll_cfg.multiplier              = (cgc_pll_mul_t) BSP_CFG_PLL_MUL;
    clocks_cfg.pll_cfg.divider                 = (cgc_pll_div_t) BSP_CFG_PLL_DIV;
    clocks_cfg.divider_cfg.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
```

```
    clocks_cfg.divider_cfg.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state                   = CGC_CLOCK_CHANGE_START;
    clocks_cfg.hoco_state                      = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.moco_state                      = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.loco_state                      = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
#endif
}
```

### Oscillation Stop Detection

This example demonstrates registering a callback for oscillation stop detection of the main oscillator.

```
/* Example callback called when oscillation stop is detected. */
void oscillation_stop_callback (cgc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If the MCU was running on the main oscillator, the MCU is now running
on MOCO. Switch clocks if
  * desired. This example shows switching to HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for HOCO to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t dividers =
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
       .sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
```

```
      .sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4,
      .sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4,
      .sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4,
      .sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4,
#else
 #if BSP_FEATURE_CGC_HAS_MOCO
      .moco_divider         = CGC_SYS_CLOCK_DIV_2,
 #endif
      .hoco_divider         = CGC_SYS_CLOCK_DIV_1,
 #if BSP_FEATURE_CGC_HAS_MOSC
      .mosc_divider         = CGC_SYS_CLOCK_DIV_4,
 #endif
#endif
   };
   err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_HOCO, &dividers);
   assert(FSP_SUCCESS == err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* (Optional) If the MCU was running on the PLL, the PLL is now in free-running
mode. Switch clocks if
  * desired. This example shows switching to the PLL running on HOCO. */
   err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
   assert(FSP_SUCCESS == err);
 do
   {
 /* Wait for HOCO to stabilize. */
      err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
   } while (FSP_SUCCESS != err);
 cgc_pll_cfg_t pll_cfg =
   {
      .source_clock = CGC_CLOCK_HOCO,
      .multiplier   = (cgc_pll_mul_t) BSP_CFG_PLL_MUL,
      .divider      = (cgc_pll_div_t) BSP_CFG_PLL_DIV,
   };
   err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_cfg);
```

```
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for PLL to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_PLL);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t pll_dividers =
    {
       .sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4,
       .sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_dividers);
    assert(FSP_SUCCESS == err);
#endif
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
void cgc_osc_stop_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable oscillation stop detection. The main oscillator must be running at this
point. */
```

```
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WFI();
 /* (Optional) Reenable oscillation stop detection after waking from low power mode.
*/
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | cgc_instance_ctrl_t |

## Data Structure Documentation

### ◆ cgc_instance_ctrl_t

| struct cgc_instance_ctrl_t | |
|---|---|
| CGC private control block. DO NOT MODIFY. Initialization occurs when R_CGC_Open() is called. | |
| **Data Fields** | |
| void * | p_context |

## Field Documentation

### ◆ p_context

| void* cgc_instance_ctrl_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in cgc_callback_args_t. |

## Function Documentation

### ◆ R_CGC_Open()

fsp_err_t R_CGC_Open ( cgc_ctrl_t *const *p_ctrl*, cgc_cfg_t const *const *p_cfg* )

Initialize the CGC API. Implements cgc_api_t::open.

Example:

```
/* Initializes the CGC module. */
   err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | CGC successfully initialized. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_CGC_ClocksCfg()

fsp_err_t R_CGC_ClocksCfg ( cgc_ctrl_t *const *p_ctrl*, cgc_clocks_cfg_t const *const *p_clock_cfg* )

Reconfigures all main system clocks. This API can be used for any of the following purposes:

- start or stop clocks
- change the system clock source
- configure the PLL/PLL2 multiplication and division ratios when starting the PLL
- configure the division ratios when starting the HOCO/MOCO/MOSC
- change the system dividers

If the requested system clock source has a stabilization flag, this function blocks waiting for the stabilization flag of the requested system clock source to be set. If the requested system clock source was just started and it has no stabilization flag, this function blocks for the stabilization time required by the requested system clock source according to the Electrical Characteristics section of the hardware manual. If the requested system clock source has no stabilization flag and it is already running, it is assumed to be stable and this function will not block. If the requested system clock is the subclock, the subclock must be stable prior to calling this function.

The internal dividers (cgc_clocks_cfg_t::divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of table "Specifications of the clock generation circuit for the internal clocks" in the CGC section of the relevant hardware manual.
- RA2A1: see footnotes of table "Clock generation circuit specifications for the internal clocks" in the CGC section of the relevant hardware manual.

Do not attempt to stop the requested clock source or the source of a PLL if the PLL will be running after this operation completes.

Implements cgc_api_t::clocksCfg.

Example:

```
/* Assuming the system clock is MOCO, switch to HOCO. */
```

```
 cgc_clocks_cfg_t clocks_cfg;
#if !BSP_FEATURE_CGC_REGISTER_SET_B
    clocks_cfg.system_clock                     = CGC_CLOCK_PLL;
    clocks_cfg.pll_state                        = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock             = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier               = CGC_PLL_MUL_10_0;   // unused
    clocks_cfg.pll_cfg.divider                  = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_4;
#else
    clocks_cfg.system_clock = CGC_CLOCK_HOCO;
 #if BSP_FEATURE_CGC_HAS_MOCO
    clocks_cfg.divider_cfg.moco_divider = CGC_SYS_CLOCK_DIV_4;
 #endif
    clocks_cfg.divider_cfg.hoco_divider = CGC_SYS_CLOCK_DIV_2;
 #if BSP_FEATURE_CGC_HAS_MOSC
    clocks_cfg.divider_cfg.mosc_divider = CGC_SYS_CLOCK_DIV_1;
 #endif
#endif
    clocks_cfg.mainosc_state = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state    = CGC_CLOCK_CHANGE_START;
#if BSP_FEATURE_CGC_HAS_MOCO
    clocks_cfg.moco_state = CGC_CLOCK_CHANGE_STOP;
#else
    clocks_cfg.moco_state = CGC_CLOCK_CHANGE_NONE;
#endif
    clocks_cfg.loco_state = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock configuration applied successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | Attempt to stop the current system clock or the PLL source clock. |
| FSP_ERR_CLOCK_ACTIVE | PLL configuration cannot be changed while PLL is running. |
| FSP_ERR_OSC_STOP_DET_ENABLED | PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz. |
| FSP_ERR_NOT_STABILIZED | PLL clock source is not stable. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL clock source is not running. |
| FSP_ERR_INVALID_STATE | The subclock must be running before activating HOCO with FLL. |

### ◆ R_CGC_ClockStart()

fsp_err_t R_CGC_ClockStart ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_pll_cfg_t const *const *p_pll_cfg* )

Start the specified clock if it is not currently active. The PLL configuration cannot be changed while the PLL is running. Implements cgc_api_t::clockStart.

The PLL source clock must be operating and stable prior to starting the PLL.

Example:

```
/* Start the LOCO. */
   err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock initialized successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_STABILIZED | The clock source is not stabilized after being turned off or PLL clock source is not stable. |
| FSP_ERR_PLL_SRC_INACTIVE | PLL clock source is not running. |
| FSP_ERR_CLOCK_ACTIVE | PLL configuration cannot be changed while PLL is running. |
| FSP_ERR_OSC_STOP_DET_ENABLED | PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz. |
| FSP_ERR_INVALID_STATE | The subclock must be running before activating HOCO with FLL. |

### ◆ R_CGC_ClockStop()

fsp_err_t R_CGC_ClockStop ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source* )

Stop the specified clock if it is active. Implements cgc_api_t::clockStop.

Do not attempt to stop the current system clock source. Do not attempt to stop the source clock of a PLL if the PLL is running.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock stopped successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_IN_USE | Attempt to stop the current system clock or the PLL source clock. |
| FSP_ERR_OSC_STOP_DET_ENABLED | Attempt to stop MOCO when Oscillation stop is enabled. |
| FSP_ERR_NOT_STABILIZED | Clock not stabilized after starting. |

### ◆ R_CGC_SystemClockSet()

fsp_err_t R_CGC_SystemClockSet ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_divider_cfg_t const *const *p_divider_cfg* )

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD, BCLK, and FCLK. Implements cgc_api_t::systemClockSet.

The requested clock source must be running and stable prior to calling this function. The internal dividers are subject to constraints described in the hardware manual table "Specifications of the Clock Generation Circuit for the internal clocks".

The internal dividers (p_divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes "Specifications of the clock generation circuit for the internal clocks" in the CGC section of the relevant hardware manual.
- RA2A1: see footnotes "Clock generation circuit specifications for the internal clocks" in the CGC section of the relevant hardware manual.

This function also updates the RAM and ROM wait states, the operating power control mode, and the SystemCoreClock CMSIS global variable.

Example:

```
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
#if !BSP_FEATURE_CGC_REGISTER_SET_B
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_b.pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
       .sckdivcr_b.pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .sckdivcr_b.iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .sckdivcr_b.pclka_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.fclk_div = CGC_SYS_CLOCK_DIV_1,
       .sckdivcr_b.bclk_div = CGC_SYS_CLOCK_DIV_1,
#else
```

```
#if BSP_FEATURE_CGC_HAS_MOCO
      .moco_divider         = CGC_SYS_CLOCK_DIV_2,
#endif
      .hoco_divider         = CGC_SYS_CLOCK_DIV_1,
#if BSP_FEATURE_CGC_HAS_MOSC
      .mosc_divider         = CGC_SYS_CLOCK_DIV_4,
#endif
#endif
   };
/* Switch the system clock to LOCO. */
   err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CLOCK_INACTIVE | The specified clock source is inactive. |
| FSP_ERR_NOT_STABILIZED | The clock source has not stabilized |

### ◆ R_CGC_SystemClockGet()

fsp_err_t R_CGC_SystemClockGet ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *const *p_clock_source*, cgc_divider_cfg_t *const *p_divider_cfg* )

Return the current system clock source and configuration. Implements cgc_api_t::systemClockGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Parameters returned successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |

### ◆ R_CGC_ClockCheck()

fsp_err_t R_CGC_ClockCheck ( cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source* )

Check the specified clock for stability. Implements cgc_api_t::clockCheck.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Clock is running and stable. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_NOT_STABILIZED | Clock not stabilized. |
| FSP_ERR_CLOCK_INACTIVE | Clock not turned on. |

### ◆ R_CGC_OscStopDetectEnable()

fsp_err_t R_CGC_OscStopDetectEnable ( cgc_ctrl_t *const *p_ctrl*)

Enable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectEnable.

The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency.

Example:

```
/* Enable oscillation stop detection. The main oscillator must be running at this
point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_LOW_VOLTAGE_MODE | Settings not allowed in low voltage mode. |
| FSP_ERR_UNSUPPORTED | Function not supported. |

### ◆ R_CGC_OscStopDetectDisable()

fsp_err_t R_CGC_OscStopDetectDisable ( cgc_ctrl_t *const *p_ctrl*)

Disable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectDisable.

Example:

```
/* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WFI();
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_OSC_STOP_DETECTED | The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function. |
| FSP_ERR_UNSUPPORTED | Function not supported. |

### ◆ R_CGC_OscStopStatusClear()

fsp_err_t R_CGC_OscStopStatusClear ( cgc_ctrl_t *const *p_ctrl*)

Clear the Oscillation Stop Detection Status register. This register is not cleared automatically if the stopped clock is restarted. Implements cgc_api_t::oscStopStatusClear.

After clearing the status, oscillation stop detection is no longer enabled.

This register cannot be cleared while the main oscillator is the system clock or the PLL source clock.

Example:

```
/* (Optional) Clear the error flag. Only clear this flag after switching the MCU clock source away from the main
 * oscillator and if the main oscillator is stable again. */
  err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
  assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Operation performed successfully. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |
| FSP_ERR_CLOCK_INACTIVE | Main oscillator must be running to clear the oscillation stop detection flag. |
| FSP_ERR_OSC_STOP_CLOCK_ACTIVE | The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit. |
| FSP_ERR_INVALID_HW_CONDITION | Oscillation stop status was not cleared. Check preconditions and try again. |
| FSP_ERR_UNSUPPORTED | Function not supported. |

### ◆ R_CGC_CallbackSet()

fsp_err_t R_CGC_CallbackSet ( cgc_ctrl_t *const *p_api_ctrl*, void(*)(cgc_callback_args_t *) *p_callback*, void *const *p_context*, cgc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements cgc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |
| FSP_ERR_UNSUPPORTED | Function not supported. |

### ◆ R_CGC_Close()

fsp_err_t R_CGC_Close ( cgc_ctrl_t *const *p_ctrl*)

Closes the CGC module. Implements cgc_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The module is successfully closed. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Module is not open. |

## 5.2.18.2 Event Link Controller (r_elc)

Modules » System

**Functions**

| | |
|---|---|
| fsp_err_t | R_ELC_Open (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg) |
| fsp_err_t | R_ELC_Close (elc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ELC_SoftwareEventGenerate (elc_ctrl_t *const p_ctrl, elc_software_event_t event_number) |
| fsp_err_t | R_ELC_LinkSet (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, |

| | |
|---|---|
| | elc_event_t signal) |
| fsp_err_t | R_ELC_LinkBreak (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral) |
| fsp_err_t | R_ELC_Enable (elc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ELC_Disable (elc_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface.

# Overview

The event link controller (ELC) uses the event requests generated by various peripheral modules as source signals to connect (link) them to different modules, allowing direct cooperation between the modules without central processing unit (CPU) intervention. The conceptual diagram below illustrates a potential setup where a pin interrupt triggers a timer which later triggers an ADC conversion and CTSU scan, while at the same time a serial communication interrupt automatically starts a data transfer. These tasks would be automatically handled without the need for polling or interrupt management.

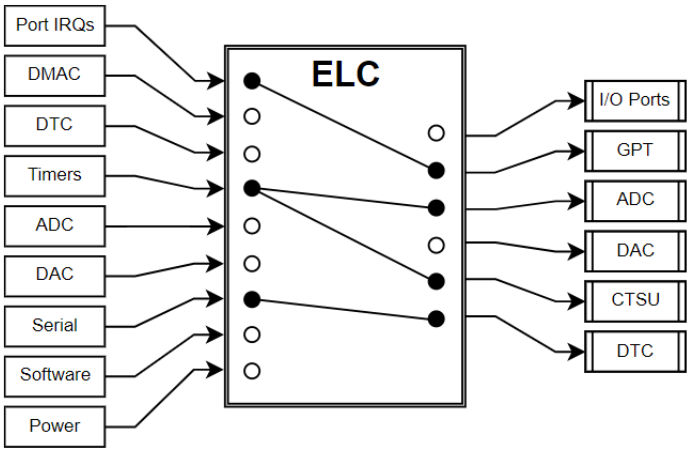



Figure 334: Event Link Controller Conceptual Diagram

In essence, the ELC is an array of multiplexers to route a wide variety of interrupt signals to a subset of peripheral functions. Events are linked by setting the multiplexer for the desired function to the desired signal (through R_ELC_LinkSet). The diagram below illustrates one peripheral output of the ELC. In this example, a conversion start is triggered for ADC0 Group A when the GPT0 counter overflows:

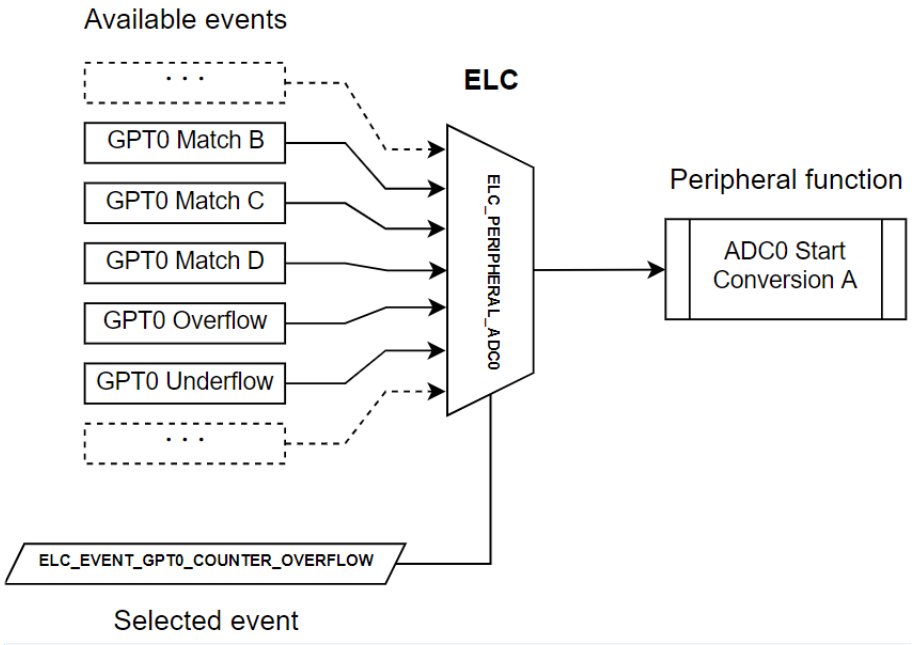



Figure 335: ELC Example

### Features

The ELC HAL module can perform the following functions:

- Initialize the ELC to a pre-defined set of links
- Create an event link between two blocks
- Break an event link between two blocks
- Generate one of two software events that interrupt the CPU
- Globally enable or disable event links

A variety of functions can be activated via events, including:

- General-purpose timer (GPT) control
- ADC and DAC conversion start
- Synchronized I/O port output (ports 1-4 only)
- Capacitive touch unit (CTSU) measurement activation

*Note*

*The available sources and peripherals may differ between devices. A full list of selectable peripherals and events is available in the User's Manual for your device.*
*Some peripherals have specific settings related to ELC event generation and/or reception. Details on how to enable event functionality for each peripheral are located in the usage notes for the related module(s) as well as in the User's Manual for your device.*

# Configuration

*Note*

*Event links will be automatically generated based on the selections made in module properties. To view the currently linked events check the Event Links tab in the RA Configuration editor.*
*Calling R_ELC_Open followed by R_ELC_Enable will automatically link all events shown in the Event Links tab.*

To manually link an event to a peripheral at runtime perform the following steps:

1. Configure the operation of the destination peripheral (including any configuration necessary to receive events)
2. Use R_ELC_LinkSet to set the desired event link to the peripheral
3. Use R_ELC_Enable to enable transmission of event signals
4. Configure the signaling module to output the desired event (typically an interrupt)

To disable the event, either use R_ELC_LinkBreak to clear the link for a specific event or R_ELC_Disable to globally disable event linking.

*Note*

*The ELC module needs no pin, clocking or interrupt configuration; it is merely a mechanism to connect signals between peripherals. However, when linking I/O Ports via the ELC the relevant I/O pins need to be configured as inputs or outputs.*

### Build Time Configurations for r_elc

The following build time configurations are defined in fsp_cfg/r_elc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for System > Event Link Controller (r_elc)

This module can be added to the Stacks tab via New Stack > System > Event Link Controller (r_elc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | ELC instance name must be g_elc to match elc_cfg_t data structure created in elc_data.c | g_elc | Module name. Fixed to g_elc. |

## Usage Notes

### Limitations

Developers should be aware of the following limitations when using the ELC:

- To link events it is necessary for the ELC and the related modules to be enabled. The ELC cannot operate if the related modules are in the module stop state or the MCU is in a low power consumption mode for which the module is stopped.
- If two modules are linked across clock domains there may be a 1 to 2 cycle delay between event signaling and reception. The delay timing is based on the frequency of the slowest clock.

## Examples

### Basic Example

Below is a basic example of minimal use of event linking in an application.

```
/* This struct is automatically generated based on the events configured by
peripherals in the RA Configuration editor. */
static const elc_cfg_t g_elc_cfg =
{
#if (BSP_CFG_MCU_PART_SERIES == 0)
    .link[ELC_PERIPHERAL_TAU01]   = ELC_EVENT_ICU_IRQ0,
    .link[ELC_PERIPHERAL_IOPORT1] = ELC_EVENT_TAU0_TMI01
#else
    .link[ELC_PERIPHERAL_GPT_A]   = ELC_EVENT_ICU_IRQ0,
    .link[ELC_PERIPHERAL_IOPORT1] = ELC_EVENT_GPT4_COUNTER_OVERFLOW
#endif
};
void elc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Create or modify a link between a peripheral function and an event source. */
#if (BSP_CFG_MCU_PART_SERIES == 0)
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_AD, ELC_EVENT_TAU0_TMI01);
#else
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
#endif
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### Software-Generated Events

This example demonstrates how to use a software-generated event to signal a peripheral. This can be useful when the desired event source is not supported by the ELC hardware.

```
/* Interrupt handler for peripheral event not supported by the ELC */
void peripheral_isr (void)
{
 fsp_err_t err;
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
}
void elc_software_event (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Link ADC0 conversion start to software event 0. */
#if (BSP_CFG_MCU_PART_SERIES == 0)
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_AD,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
#else
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
#endif
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application code here. */
    }
}
```

### Data Structures

| | |
|---|---|
| struct | elc_instance_ctrl_t |

## Data Structure Documentation

### ◆ elc_instance_ctrl_t

| struct elc_instance_ctrl_t |
|---|
| ELC private control block. DO NOT MODIFY. Initialization occurs when R_ELC_Open() is called. |

## Function Documentation

### ◆ R_ELC_Open()

fsp_err_t R_ELC_Open ( elc_ctrl_t *const *p_ctrl*, elc_cfg_t const *const *p_cfg* )

Initialize all the links in the Event Link Controller. Implements elc_api_t::open

The configuration structure passed in to this function includes links for every event source included in the ELC and sets them all at once. To set or clear an individual link use R_ELC_LinkSet and R_ELC_LinkBreak respectively.

Example:

```
/* Initializes the software and sets the links defined in the control structure. */
   err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | p_ctrl or p_cfg was NULL |
| FSP_ERR_ALREADY_OPEN | The module is currently open |

### ◆ R_ELC_Close()

fsp_err_t R_ELC_Close ( elc_ctrl_t *const *p_ctrl*)

Globally disable ELC linking. Implements elc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The ELC was successfully disabled |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_SoftwareEventGenerate()

fsp_err_t R_ELC_SoftwareEventGenerate ( elc_ctrl_t *const *p_ctrl*, elc_software_event_t *event_number* )

Generate a software event in the Event Link Controller. Implements elc_api_t::softwareEventGenerate

Example:

```
/* Generate an event signal through software to the linked peripheral. */
   err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | Invalid event number or p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_LinkSet()

fsp_err_t R_ELC_LinkSet ( elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*, elc_event_t *signal* )

Create a single event link. Implements elc_api_t::linkSet

Example:

```
 /* Create or modify a link between a peripheral function and an event source. */
#if (BSP_CFG_MCU_PART_SERIES == 0)
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_AD, ELC_EVENT_TAU0_TMI01);
#else
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
#endif
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_LinkBreak()

fsp_err_t R_ELC_LinkBreak ( elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral* )

Break an event link. Implements elc_api_t::linkBreak

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Event link broken |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_Enable()

fsp_err_t R_ELC_Enable ( elc_ctrl_t *const *p_ctrl*)

Enable the operation of the Event Link Controller. Implements elc_api_t::enable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ELC enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_ELC_Disable()

fsp_err_t R_ELC_Disable ( elc_ctrl_t *const *p_ctrl*)

Disable the operation of the Event Link Controller. Implements elc_api_t::disable

**Return values**

| | |
|---|---|
| FSP_SUCCESS | ELC disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

## 5.2.18.3 I/O Port (r_ioport)

Modules » System

**Functions**

| | |
|---|---|
| fsp_err_t | R_IOPORT_Open (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |

| | |
|---|---|
| fsp_err_t | R_IOPORT_Close (ioport_ctrl_t *const p_ctrl) |
| fsp_err_t | R_IOPORT_PinsCfg (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| fsp_err_t | R_IOPORT_PinCfg (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg) |
| fsp_err_t | R_IOPORT_PinRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value) |
| fsp_err_t | R_IOPORT_PortRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value) |
| fsp_err_t | R_IOPORT_PortWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask) |
| fsp_err_t | R_IOPORT_PinWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level) |
| fsp_err_t | R_IOPORT_PortDirectionSet (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask) |
| fsp_err_t | R_IOPORT_PortEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data) |
| fsp_err_t | R_IOPORT_PinEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event) |
| fsp_err_t | R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value) |
| fsp_err_t | R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value) |

### Detailed Description

Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface.

# Overview

The I/O port pins operate as general I/O port pins, I/O pins for peripheral modules, interrupt input pins, analog I/O, port group function for the ELC, or bus control pins.

### Features

The IOPORT HAL module can configure the following pin settings:

- Pin direction
- Default output state
- Pull-up
- NMOS/PMOS
- Drive strength
- Event edge trigger (falling, rising or both)
- Whether the pin is to be used as an IRQ pin
- Whether the pin is to be used as an analog pin
- Peripheral connection

The module also provides the following functionality:

- Read/write GPIO pins/ports
- Sets event output data
- Reads event input data
- Controlled Current Drive (CCD)

# Configuration

The I/O PORT HAL module must be configured by the user for the desired operation. The operating state of an I/O pin can be set via the RA Configuration tool. When the project is built a pin configuration file is created. The BSP will automatically configure the MCU IO ports accordingly at startup using the same API functions mentioned in this document.

### Build Time Configurations for r_ioport

The following build time configurations are defined in fsp_cfg/r_ioport_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| CCD Support | MCU Specific Options | | Enable support for CCD (Controlled Current Drive) port pins. |

### Configurations for System > I/O Port (r_ioport)

This module can be added to the Stacks tab via New Stack > System > I/O Port (r_ioport).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ioport | Module name. |
| 1st Port ELC Trigger Source | MCU Specific Options | | ELC source that will trigger the 1st port |
| 2nd Port ELC Trigger Source | MCU Specific Options | | ELC source that will trigger the 2nd port |
| 3rd Port ELC Trigger Source | MCU Specific Options | | ELC source that will trigger the 3rd port |

| | | | |
|---|---|---|---|
| 4th Port ELC Trigger Source | MCU Specific Options | | ELC source that will trigger the 4th port |
| Pin Configuration Name | Name must be a valid C symbol | g_bsp_pin_cfg | Name for pin configuration structure |

### Clock Configuration

The I/O PORT HAL module does not require a specific clock configuration.

### Pin Configuration

The IOPORT module is used for configuring pins.

# Usage Notes

### Port Group Function for ELC

Depending on pin configuration, the IOPORT module can perform automatic reads and writes on 4 ports (RA2 and RA0 series support 2 ports only, see the table of ELC triggers below) on receipt of an ELC event.

| ELC triggers | RA6T2 | RA2XX, RA0XX (with ELC) | RA0XX (without ELC) | All others |
|---|---|---|---|---|
| 1st Port | Port B | Port 1 | NA | Port 1 |
| 2nd Port | Port C | Port 2 | NA | Port 2 |
| 3rd Port | Port D | NA | NA | Port 3 |
| 4th Port | Port E | NA | NA | Port 4 |

When an event is received by a port, the state of the input pins on the port is saved in a hardware register. Simultaneously, the state of output pins on the port is set or cleared based on settings configured by the user. The functions R_IOPORT_PinEventInputRead and R_IOPORT_PortEventInputRead allow reading the last event input state of a pin or port, and event-triggered pin output can be configured through R_IOPORT_PinEventOutputWrite and R_IOPORT_PortEventOutputWrite.

In addition, each pin on these ports can be configured to trigger an ELC event on rising, falling or both edges. This event can be used to activate other modules when the pin changes state.

*Note*

*The number of ELC-aware ports vary across MCUs. Refer to the Hardware User's Manual for your device for more details.*

# Examples

### Basic Example

This is a basic example of minimal use of the IOPORT in an application.

```
void basic_example ()
```

```
{
 bsp_io_level_t readLevel;
 fsp_err_t      err;
 /* Initialize the IOPORT module and configure the pins
  * Note: The default pin configuration name in the RA Configuration tool is
g_bsp_pin_cfg */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Call R_IOPORT_PinsCfg if the configuration was not part of initial configurations
made in open */
    err = R_IOPORT_PinsCfg(&g_ioport_ctrl, &g_runtime_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Set Pin 00 of Port 06 to High */
    err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, BSP_IO_LEVEL_HIGH
);
    assert(FSP_SUCCESS == err);
 /* Read Pin 00 of Port 06*/
    err = R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, &readLevel);
    assert(FSP_SUCCESS == err);
}
```

### Blinky Example

This example uses IOPORT to configure and toggle a pin to blink an LED.

```
void blinky_example ()
{
 fsp_err_t err;
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure Pin as output
  * Call the R_IOPORT_PinCfg if the configuration was not part of initial
```

```
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00,
IOPORT_CFG_PORT_DIRECTION_OUTPUT);
    assert(FSP_SUCCESS == err);
 bsp_io_level_t level = BSP_IO_LEVEL_LOW;
 while (1)
    {
 /* Determine the next state of the LEDs */
 if (BSP_IO_LEVEL_LOW == level)
     {
           level = BSP_IO_LEVEL_HIGH;
      }
 else
     {
           level = BSP_IO_LEVEL_LOW;
      }
 /* Update LED on RA6M3-PK */
       err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, level);
     assert(FSP_SUCCESS == err);
 /* Delay */
 R_BSP_SoftwareDelay(100, BSP_DELAY_UNITS_MILLISECONDS); // NOLINT
    }
}
```

### ELC Example

This is an example of using IOPORT with ELC events. The ELC event system allows the captured data to be stored when it occurs and then read back at a later time.

```
static elc_instance_ctrl_t g_elc_ctrl;
static elc_cfg_t g_elc_cfg;
void ioport_elc_example ()
{
 bsp_io_level_t eventValue;
 fsp_err_t      err;
```

```
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, (elc_peripheral_t) ELC_PERIPHERAL_IOPORT2,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
IOPORT_CFG_PORT_DIRECTION_INPUT);
    assert(FSP_SUCCESS == err);
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 /* Read Pin Event Input. The data(BSP_IO_LEVEL_HIGH/ BSP_IO_LEVEL_LOW) from
BSP_IO_PORT_02_PIN_00 is read into the
  * EIDR bit */
    err = R_IOPORT_PinEventInputRead(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
&eventValue);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | ioport_instance_ctrl_t |
| struct | ioport_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | ioport_port_pin_t |
| enum | ioport_peripheral_t |
| enum | ioport_cfg_options_t |
| enum | ioport_output_current_t |

## Data Structure Documentation

### ◆ ioport_instance_ctrl_t

| struct ioport_instance_ctrl_t |
|---|
| IOPORT private control block. DO NOT MODIFY. Initialization occurs when R_IOPORT_Open() is called. |

### ◆ ioport_extended_cfg_t

| struct ioport_extended_cfg_t | | |
|---|---|---|
| R_IOPORT extended configuration | | |
| Data Fields | | |
| ioport_output_current_t const | ccd_pin_cfg_data[IOPORT_PRV_CCD_PIN_COUNT] | Low-level output current for the CCD pins. |

## Enumeration Type Documentation

### ◆ ioport_port_pin_t

| enum ioport_port_pin_t | |
|---|---|
| Superset list of all possible IO port pins. | |
| Enumerator | |
| IOPORT_PORT_00_PIN_00 | IO port 0 pin 0. |
| IOPORT_PORT_00_PIN_01 | IO port 0 pin 1. |
| IOPORT_PORT_00_PIN_02 | IO port 0 pin 2. |
| IOPORT_PORT_00_PIN_03 | IO port 0 pin 3. |
| IOPORT_PORT_00_PIN_04 | IO port 0 pin 4. |
| IOPORT_PORT_00_PIN_05 | IO port 0 pin 5. |
| IOPORT_PORT_00_PIN_06 | IO port 0 pin 6. |
| IOPORT_PORT_00_PIN_07 | |

| | |
|---|---|
| | IO port 0 pin 7. |
| IOPORT_PORT_00_PIN_08 | IO port 0 pin 8. |
| IOPORT_PORT_00_PIN_09 | IO port 0 pin 9. |
| IOPORT_PORT_00_PIN_10 | IO port 0 pin 10. |
| IOPORT_PORT_00_PIN_11 | IO port 0 pin 11. |
| IOPORT_PORT_00_PIN_12 | IO port 0 pin 12. |
| IOPORT_PORT_00_PIN_13 | IO port 0 pin 13. |
| IOPORT_PORT_00_PIN_14 | IO port 0 pin 14. |
| IOPORT_PORT_00_PIN_15 | IO port 0 pin 15. |
| IOPORT_PORT_01_PIN_00 | IO port 1 pin 0. |
| IOPORT_PORT_01_PIN_01 | IO port 1 pin 1. |
| IOPORT_PORT_01_PIN_02 | IO port 1 pin 2. |
| IOPORT_PORT_01_PIN_03 | IO port 1 pin 3. |
| IOPORT_PORT_01_PIN_04 | IO port 1 pin 4. |
| IOPORT_PORT_01_PIN_05 | IO port 1 pin 5. |
| IOPORT_PORT_01_PIN_06 | IO port 1 pin 6. |
| IOPORT_PORT_01_PIN_07 | IO port 1 pin 7. |
| IOPORT_PORT_01_PIN_08 | IO port 1 pin 8. |
| IOPORT_PORT_01_PIN_09 | IO port 1 pin 9. |
| IOPORT_PORT_01_PIN_10 | IO port 1 pin 10. |
| IOPORT_PORT_01_PIN_11 | IO port 1 pin 11. |
| IOPORT_PORT_01_PIN_12 | IO port 1 pin 12. |
| IOPORT_PORT_01_PIN_13 | IO port 1 pin 13. |
| IOPORT_PORT_01_PIN_14 | IO port 1 pin 14. |
| IOPORT_PORT_01_PIN_15 | |

| | |
|---|---|
| | IO port 1 pin 15. |
| IOPORT_PORT_02_PIN_00 | IO port 2 pin 0. |
| IOPORT_PORT_02_PIN_01 | IO port 2 pin 1. |
| IOPORT_PORT_02_PIN_02 | IO port 2 pin 2. |
| IOPORT_PORT_02_PIN_03 | IO port 2 pin 3. |
| IOPORT_PORT_02_PIN_04 | IO port 2 pin 4. |
| IOPORT_PORT_02_PIN_05 | IO port 2 pin 5. |
| IOPORT_PORT_02_PIN_06 | IO port 2 pin 6. |
| IOPORT_PORT_02_PIN_07 | IO port 2 pin 7. |
| IOPORT_PORT_02_PIN_08 | IO port 2 pin 8. |
| IOPORT_PORT_02_PIN_09 | IO port 2 pin 9. |
| IOPORT_PORT_02_PIN_10 | IO port 2 pin 10. |
| IOPORT_PORT_02_PIN_11 | IO port 2 pin 11. |
| IOPORT_PORT_02_PIN_12 | IO port 2 pin 12. |
| IOPORT_PORT_02_PIN_13 | IO port 2 pin 13. |
| IOPORT_PORT_02_PIN_14 | IO port 2 pin 14. |
| IOPORT_PORT_02_PIN_15 | IO port 2 pin 15. |
| IOPORT_PORT_03_PIN_00 | IO port 3 pin 0. |
| IOPORT_PORT_03_PIN_01 | IO port 3 pin 1. |
| IOPORT_PORT_03_PIN_02 | IO port 3 pin 2. |
| IOPORT_PORT_03_PIN_03 | IO port 3 pin 3. |
| IOPORT_PORT_03_PIN_04 | IO port 3 pin 4. |
| IOPORT_PORT_03_PIN_05 | IO port 3 pin 5. |
| IOPORT_PORT_03_PIN_06 | IO port 3 pin 6. |
| IOPORT_PORT_03_PIN_07 | |

| | |
|---|---|
| | IO port 3 pin 7. |
| IOPORT_PORT_03_PIN_08 | IO port 3 pin 8. |
| IOPORT_PORT_03_PIN_09 | IO port 3 pin 9. |
| IOPORT_PORT_03_PIN_10 | IO port 3 pin 10. |
| IOPORT_PORT_03_PIN_11 | IO port 3 pin 11. |
| IOPORT_PORT_03_PIN_12 | IO port 3 pin 12. |
| IOPORT_PORT_03_PIN_13 | IO port 3 pin 13. |
| IOPORT_PORT_03_PIN_14 | IO port 3 pin 14. |
| IOPORT_PORT_03_PIN_15 | IO port 3 pin 15. |
| IOPORT_PORT_04_PIN_00 | IO port 4 pin 0. |
| IOPORT_PORT_04_PIN_01 | IO port 4 pin 1. |
| IOPORT_PORT_04_PIN_02 | IO port 4 pin 2. |
| IOPORT_PORT_04_PIN_03 | IO port 4 pin 3. |
| IOPORT_PORT_04_PIN_04 | IO port 4 pin 4. |
| IOPORT_PORT_04_PIN_05 | IO port 4 pin 5. |
| IOPORT_PORT_04_PIN_06 | IO port 4 pin 6. |
| IOPORT_PORT_04_PIN_07 | IO port 4 pin 7. |
| IOPORT_PORT_04_PIN_08 | IO port 4 pin 8. |
| IOPORT_PORT_04_PIN_09 | IO port 4 pin 9. |
| IOPORT_PORT_04_PIN_10 | IO port 4 pin 10. |
| IOPORT_PORT_04_PIN_11 | IO port 4 pin 11. |
| IOPORT_PORT_04_PIN_12 | IO port 4 pin 12. |
| IOPORT_PORT_04_PIN_13 | IO port 4 pin 13. |
| IOPORT_PORT_04_PIN_14 | IO port 4 pin 14. |
| IOPORT_PORT_04_PIN_15 | |

| | |
|---|---|
| | IO port 4 pin 15. |
| IOPORT_PORT_05_PIN_00 | IO port 5 pin 0. |
| IOPORT_PORT_05_PIN_01 | IO port 5 pin 1. |
| IOPORT_PORT_05_PIN_02 | IO port 5 pin 2. |
| IOPORT_PORT_05_PIN_03 | IO port 5 pin 3. |
| IOPORT_PORT_05_PIN_04 | IO port 5 pin 4. |
| IOPORT_PORT_05_PIN_05 | IO port 5 pin 5. |
| IOPORT_PORT_05_PIN_06 | IO port 5 pin 6. |
| IOPORT_PORT_05_PIN_07 | IO port 5 pin 7. |
| IOPORT_PORT_05_PIN_08 | IO port 5 pin 8. |
| IOPORT_PORT_05_PIN_09 | IO port 5 pin 9. |
| IOPORT_PORT_05_PIN_10 | IO port 5 pin 10. |
| IOPORT_PORT_05_PIN_11 | IO port 5 pin 11. |
| IOPORT_PORT_05_PIN_12 | IO port 5 pin 12. |
| IOPORT_PORT_05_PIN_13 | IO port 5 pin 13. |
| IOPORT_PORT_05_PIN_14 | IO port 5 pin 14. |
| IOPORT_PORT_05_PIN_15 | IO port 5 pin 15. |
| IOPORT_PORT_06_PIN_00 | IO port 6 pin 0. |
| IOPORT_PORT_06_PIN_01 | IO port 6 pin 1. |
| IOPORT_PORT_06_PIN_02 | IO port 6 pin 2. |
| IOPORT_PORT_06_PIN_03 | IO port 6 pin 3. |
| IOPORT_PORT_06_PIN_04 | IO port 6 pin 4. |
| IOPORT_PORT_06_PIN_05 | IO port 6 pin 5. |
| IOPORT_PORT_06_PIN_06 | IO port 6 pin 6. |
| IOPORT_PORT_06_PIN_07 | |

| | |
|---|---|
| | IO port 6 pin 7. |
| IOPORT_PORT_06_PIN_08 | IO port 6 pin 8. |
| IOPORT_PORT_06_PIN_09 | IO port 6 pin 9. |
| IOPORT_PORT_06_PIN_10 | IO port 6 pin 10. |
| IOPORT_PORT_06_PIN_11 | IO port 6 pin 11. |
| IOPORT_PORT_06_PIN_12 | IO port 6 pin 12. |
| IOPORT_PORT_06_PIN_13 | IO port 6 pin 13. |
| IOPORT_PORT_06_PIN_14 | IO port 6 pin 14. |
| IOPORT_PORT_06_PIN_15 | IO port 6 pin 15. |
| IOPORT_PORT_07_PIN_00 | IO port 7 pin 0. |
| IOPORT_PORT_07_PIN_01 | IO port 7 pin 1. |
| IOPORT_PORT_07_PIN_02 | IO port 7 pin 2. |
| IOPORT_PORT_07_PIN_03 | IO port 7 pin 3. |
| IOPORT_PORT_07_PIN_04 | IO port 7 pin 4. |
| IOPORT_PORT_07_PIN_05 | IO port 7 pin 5. |
| IOPORT_PORT_07_PIN_06 | IO port 7 pin 6. |
| IOPORT_PORT_07_PIN_07 | IO port 7 pin 7. |
| IOPORT_PORT_07_PIN_08 | IO port 7 pin 8. |
| IOPORT_PORT_07_PIN_09 | IO port 7 pin 9. |
| IOPORT_PORT_07_PIN_10 | IO port 7 pin 10. |
| IOPORT_PORT_07_PIN_11 | IO port 7 pin 11. |
| IOPORT_PORT_07_PIN_12 | IO port 7 pin 12. |
| IOPORT_PORT_07_PIN_13 | IO port 7 pin 13. |
| IOPORT_PORT_07_PIN_14 | IO port 7 pin 14. |
| IOPORT_PORT_07_PIN_15 | |

| | |
|---|---|
| | IO port 7 pin 15. |
| IOPORT_PORT_08_PIN_00 | IO port 8 pin 0. |
| IOPORT_PORT_08_PIN_01 | IO port 8 pin 1. |
| IOPORT_PORT_08_PIN_02 | IO port 8 pin 2. |
| IOPORT_PORT_08_PIN_03 | IO port 8 pin 3. |
| IOPORT_PORT_08_PIN_04 | IO port 8 pin 4. |
| IOPORT_PORT_08_PIN_05 | IO port 8 pin 5. |
| IOPORT_PORT_08_PIN_06 | IO port 8 pin 6. |
| IOPORT_PORT_08_PIN_07 | IO port 8 pin 7. |
| IOPORT_PORT_08_PIN_08 | IO port 8 pin 8. |
| IOPORT_PORT_08_PIN_09 | IO port 8 pin 9. |
| IOPORT_PORT_08_PIN_10 | IO port 8 pin 10. |
| IOPORT_PORT_08_PIN_11 | IO port 8 pin 11. |
| IOPORT_PORT_08_PIN_12 | IO port 8 pin 12. |
| IOPORT_PORT_08_PIN_13 | IO port 8 pin 13. |
| IOPORT_PORT_08_PIN_14 | IO port 8 pin 14. |
| IOPORT_PORT_08_PIN_15 | IO port 8 pin 15. |
| IOPORT_PORT_09_PIN_00 | IO port 9 pin 0. |
| IOPORT_PORT_09_PIN_01 | IO port 9 pin 1. |
| IOPORT_PORT_09_PIN_02 | IO port 9 pin 2. |
| IOPORT_PORT_09_PIN_03 | IO port 9 pin 3. |
| IOPORT_PORT_09_PIN_04 | IO port 9 pin 4. |
| IOPORT_PORT_09_PIN_05 | IO port 9 pin 5. |
| IOPORT_PORT_09_PIN_06 | IO port 9 pin 6. |
| IOPORT_PORT_09_PIN_07 | |

| | |
|---|---|
| | IO port 9 pin 7. |
| IOPORT_PORT_09_PIN_08 | IO port 9 pin 8. |
| IOPORT_PORT_09_PIN_09 | IO port 9 pin 9. |
| IOPORT_PORT_09_PIN_10 | IO port 9 pin 10. |
| IOPORT_PORT_09_PIN_11 | IO port 9 pin 11. |
| IOPORT_PORT_09_PIN_12 | IO port 9 pin 12. |
| IOPORT_PORT_09_PIN_13 | IO port 9 pin 13. |
| IOPORT_PORT_09_PIN_14 | IO port 9 pin 14. |
| IOPORT_PORT_09_PIN_15 | IO port 9 pin 15. |
| IOPORT_PORT_10_PIN_00 | IO port 10 pin 0. |
| IOPORT_PORT_10_PIN_01 | IO port 10 pin 1. |
| IOPORT_PORT_10_PIN_02 | IO port 10 pin 2. |
| IOPORT_PORT_10_PIN_03 | IO port 10 pin 3. |
| IOPORT_PORT_10_PIN_04 | IO port 10 pin 4. |
| IOPORT_PORT_10_PIN_05 | IO port 10 pin 5. |
| IOPORT_PORT_10_PIN_06 | IO port 10 pin 6. |
| IOPORT_PORT_10_PIN_07 | IO port 10 pin 7. |
| IOPORT_PORT_10_PIN_08 | IO port 10 pin 8. |
| IOPORT_PORT_10_PIN_09 | IO port 10 pin 9. |
| IOPORT_PORT_10_PIN_10 | IO port 10 pin 10. |
| IOPORT_PORT_10_PIN_11 | IO port 10 pin 11. |
| IOPORT_PORT_10_PIN_12 | IO port 10 pin 12. |
| IOPORT_PORT_10_PIN_13 | IO port 10 pin 13. |
| IOPORT_PORT_10_PIN_14 | IO port 10 pin 14. |
| IOPORT_PORT_10_PIN_15 | |

| | |
|---|---|
| | IO port 10 pin 15. |
| IOPORT_PORT_11_PIN_00 | IO port 11 pin 0. |
| IOPORT_PORT_11_PIN_01 | IO port 11 pin 1. |
| IOPORT_PORT_11_PIN_02 | IO port 11 pin 2. |
| IOPORT_PORT_11_PIN_03 | IO port 11 pin 3. |
| IOPORT_PORT_11_PIN_04 | IO port 11 pin 4. |
| IOPORT_PORT_11_PIN_05 | IO port 11 pin 5. |
| IOPORT_PORT_11_PIN_06 | IO port 11 pin 6. |
| IOPORT_PORT_11_PIN_07 | IO port 11 pin 7. |
| IOPORT_PORT_11_PIN_08 | IO port 11 pin 8. |
| IOPORT_PORT_11_PIN_09 | IO port 11 pin 9. |
| IOPORT_PORT_11_PIN_10 | IO port 11 pin 10. |
| IOPORT_PORT_11_PIN_11 | IO port 11 pin 11. |
| IOPORT_PORT_11_PIN_12 | IO port 11 pin 12. |
| IOPORT_PORT_11_PIN_13 | IO port 11 pin 13. |
| IOPORT_PORT_11_PIN_14 | IO port 11 pin 14. |
| IOPORT_PORT_11_PIN_15 | IO port 11 pin 15. |
| IOPORT_PORT_12_PIN_00 | IO port 12 pin 0. |
| IOPORT_PORT_12_PIN_01 | IO port 12 pin 1. |
| IOPORT_PORT_12_PIN_02 | IO port 12 pin 2. |
| IOPORT_PORT_12_PIN_03 | IO port 12 pin 3. |
| IOPORT_PORT_12_PIN_04 | IO port 12 pin 4. |
| IOPORT_PORT_12_PIN_05 | IO port 12 pin 5. |
| IOPORT_PORT_12_PIN_06 | IO port 12 pin 6. |
| IOPORT_PORT_12_PIN_07 | |

| | |
|---|---|
| | IO port 12 pin 7. |
| IOPORT_PORT_12_PIN_08 | IO port 12 pin 8. |
| IOPORT_PORT_12_PIN_09 | IO port 12 pin 9. |
| IOPORT_PORT_12_PIN_10 | IO port 12 pin 10. |
| IOPORT_PORT_12_PIN_11 | IO port 12 pin 11. |
| IOPORT_PORT_12_PIN_12 | IO port 12 pin 12. |
| IOPORT_PORT_12_PIN_13 | IO port 12 pin 13. |
| IOPORT_PORT_12_PIN_14 | IO port 12 pin 14. |
| IOPORT_PORT_12_PIN_15 | IO port 12 pin 15. |
| IOPORT_PORT_13_PIN_00 | IO port 13 pin 0. |
| IOPORT_PORT_13_PIN_01 | IO port 13 pin 1. |
| IOPORT_PORT_13_PIN_02 | IO port 13 pin 2. |
| IOPORT_PORT_13_PIN_03 | IO port 13 pin 3. |
| IOPORT_PORT_13_PIN_04 | IO port 13 pin 4. |
| IOPORT_PORT_13_PIN_05 | IO port 13 pin 5. |
| IOPORT_PORT_13_PIN_06 | IO port 13 pin 6. |
| IOPORT_PORT_13_PIN_07 | IO port 13 pin 7. |
| IOPORT_PORT_13_PIN_08 | IO port 13 pin 8. |
| IOPORT_PORT_13_PIN_09 | IO port 13 pin 9. |
| IOPORT_PORT_13_PIN_10 | IO port 13 pin 10. |
| IOPORT_PORT_13_PIN_11 | IO port 13 pin 11. |
| IOPORT_PORT_13_PIN_12 | IO port 13 pin 12. |
| IOPORT_PORT_13_PIN_13 | IO port 13 pin 13. |
| IOPORT_PORT_13_PIN_14 | IO port 13 pin 14. |
| IOPORT_PORT_13_PIN_15 | |

| | |
|---|---|
| | IO port 13 pin 15. |
| IOPORT_PORT_14_PIN_00 | IO port 14 pin 0. |
| IOPORT_PORT_14_PIN_01 | IO port 14 pin 1. |
| IOPORT_PORT_14_PIN_02 | IO port 14 pin 2. |
| IOPORT_PORT_14_PIN_03 | IO port 14 pin 3. |
| IOPORT_PORT_14_PIN_04 | IO port 14 pin 4. |
| IOPORT_PORT_14_PIN_05 | IO port 14 pin 5. |
| IOPORT_PORT_14_PIN_06 | IO port 14 pin 6. |
| IOPORT_PORT_14_PIN_07 | IO port 14 pin 7. |
| IOPORT_PORT_14_PIN_08 | IO port 14 pin 8. |
| IOPORT_PORT_14_PIN_09 | IO port 14 pin 9. |
| IOPORT_PORT_14_PIN_10 | IO port 14 pin 10. |
| IOPORT_PORT_14_PIN_11 | IO port 14 pin 11. |
| IOPORT_PORT_14_PIN_12 | IO port 14 pin 12. |
| IOPORT_PORT_14_PIN_13 | IO port 14 pin 13. |
| IOPORT_PORT_14_PIN_14 | IO port 14 pin 14. |
| IOPORT_PORT_14_PIN_15 | IO port 14 pin 15. |

### ◆ ioport_peripheral_t

| enum ioport_peripheral_t | |
|---|---|
| Superset of all peripheral functions. | |
| Enumerator | |
| IOPORT_PERIPHERAL_IO | Pin will functions as an IO pin |
| IOPORT_PERIPHERAL_DEBUG | Pin will function as a DEBUG pin |
| IOPORT_PERIPHERAL_AGT | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGTW | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGT1 | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_ACMPHS | Pin will function as a CMPHS peripheral pin |
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |
| IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8 | Pin will function as an SCI peripheral DEn pin |
| IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9 | Pin will function as an SCI DEn peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CEU | Pin will function as a CEU peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_GPT2 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_GPT3 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_GPT4 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |
| IOPORT_PERIPHERAL_CAC | Pin will function as a CAC peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_OSPI | Pin will function as a OSPI peripheral pin |
| IOPORT_PERIPHERAL_CEC | Pin will function as a CEC peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT0 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT1 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_ULPT | Pin will function as a ULPT peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_MIPI | Pin will function as a MIPI DSI peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RGMII | Pin will function as a RGMII peripheral pin |
| IOPORT_PERIPHERAL_PDM | Pin will function as a PDM peripheral pin |
| IOPORT_PERIPHERAL_IO | Pin will functions as an IO pin |
| IOPORT_PERIPHERAL_DEBUG | Pin will function as a DEBUG pin |
| IOPORT_PERIPHERAL_AGT | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGTW | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGT1 | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_ACMPHS | Pin will function as a CMPHS peripheral pin |
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |
| IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8 | Pin will function as an SCI peripheral DEn pin |
| IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9 | Pin will function as an SCI DEn peripheral pin |
| | |

| | |
|---|---|
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CEU | Pin will function as a CEU peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_GPT2 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_GPT3 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_GPT4 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |
| IOPORT_PERIPHERAL_CAC | Pin will function as a CAC peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_OSPI | Pin will function as a OSPI peripheral pin |
| IOPORT_PERIPHERAL_CEC | Pin will function as a CEC peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT0 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT1 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_ULPT | Pin will function as a ULPT peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_MIPI | Pin will function as a MIPI DSI peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RGMII | Pin will function as a RGMII peripheral pin |
| IOPORT_PERIPHERAL_ESC | Pin will function as a ETHERNET slave controller peripheral pin |
| IOPORT_PERIPHERAL_PDM | Pin will function as a PDM peripheral pin |
| IOPORT_PERIPHERAL_DSMIF | Pin will function as a DSMIF peripheral pin |
| IOPORT_PERIPHERAL_IO | Pin will functions as an IO pin |
| IOPORT_PERIPHERAL_DEBUG | Pin will function as a DEBUG pin |
| IOPORT_PERIPHERAL_AGT | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGTW | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGT1 | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_ACMPHS | Pin will function as a CMPHS peripheral pin |
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8 | Pin will function as an SCI peripheral DEn pin |
| IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9 | Pin will function as an SCI DEn peripheral pin |
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CEU | Pin will function as a CEU peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_GPT2 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_GPT3 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_GPT4 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |
| IOPORT_PERIPHERAL_CAC | Pin will function as a CAC peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_OSPI | Pin will function as a OSPI peripheral pin |
| IOPORT_PERIPHERAL_CEC | Pin will function as a CEC peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT0 | Pin will function as a PGAOUT peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_PGAOUT1 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_ULPT | Pin will function as a ULPT peripheral pin |
| IOPORT_PERIPHERAL_MIPI | Pin will function as a MIPI DSI peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RGMII | Pin will function as a RGMII peripheral pin |
| IOPORT_PERIPHERAL_ESC | Pin will function as a ETHERNET slave controller peripheral pin |
| IOPORT_PERIPHERAL_PDM | Pin will function as a PDM peripheral pin |
| IOPORT_PERIPHERAL_IO | Pin will functions as an IO pin |
| IOPORT_PERIPHERAL_DEBUG | Pin will function as a DEBUG pin |
| IOPORT_PERIPHERAL_AGT | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGTW | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGT1 | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_ACMPHS | Pin will function as a CMPHS peripheral pin |
| | |

| | |
|---|---|
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |
| IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8 | Pin will function as an SCI peripheral DEn pin |
| IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9 | Pin will function as an SCI DEn peripheral pin |
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CEU | Pin will function as a CEU peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_GPT2 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_GPT3 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_GPT4 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |
| IOPORT_PERIPHERAL_CAC | Pin will function as a CAC peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_OSPI | Pin will function as a OSPI peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_CEC | Pin will function as a CEC peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT0 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT1 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_ULPT | Pin will function as a ULPT peripheral pin |
| IOPORT_PERIPHERAL_MIPI | Pin will function as a MIPI DSI peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RGMII | Pin will function as a RGMII peripheral pin |
| IOPORT_PERIPHERAL_ESC | Pin will function as a ETHERNET slave controller peripheral pin |
| IOPORT_PERIPHERAL_PDM | Pin will function as a PDM peripheral pin |
| IOPORT_PERIPHERAL_DSMIF | Pin will function as a DSMIF peripheral pin |
| IOPORT_PERIPHERAL_IO | Pin will functions as an IO pin |
| IOPORT_PERIPHERAL_DEBUG | Pin will function as a DEBUG pin |
| IOPORT_PERIPHERAL_AGT | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGTW | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_AGT1 | Pin will function as an AGT peripheral pin |
| IOPORT_PERIPHERAL_GPT0 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT1 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SCI0_2_4_6_8 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SCI1_3_5_7_9 | Pin will function as an SCI peripheral pin |
| IOPORT_PERIPHERAL_SPI | Pin will function as a SPI peripheral pin |
| IOPORT_PERIPHERAL_IIC | Pin will function as a IIC peripheral pin |
| IOPORT_PERIPHERAL_KEY | Pin will function as a KEY peripheral pin |
| IOPORT_PERIPHERAL_CLKOUT_COMP_RTC | Pin will function as a clock/comparator/RTC peripheral pin |
| IOPORT_PERIPHERAL_CAC_AD | Pin will function as a CAC/ADC peripheral pin |
| | |

| | |
|---|---|
| IOPORT_PERIPHERAL_BUS | Pin will function as a BUS peripheral pin |
| IOPORT_PERIPHERAL_CTSU | Pin will function as a CTSU peripheral pin |
| IOPORT_PERIPHERAL_ACMPHS | Pin will function as a CMPHS peripheral pin |
| IOPORT_PERIPHERAL_LCDC | Pin will function as a segment LCD peripheral pin |
| IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8 | Pin will function as an SCI peripheral DEn pin |
| IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9 | Pin will function as an SCI DEn peripheral pin |
| IOPORT_PERIPHERAL_DALI | Pin will function as a DALI peripheral pin |
| IOPORT_PERIPHERAL_CEU | Pin will function as a CEU peripheral pin |
| IOPORT_PERIPHERAL_CAN | Pin will function as a CAN peripheral pin |
| IOPORT_PERIPHERAL_QSPI | Pin will function as a QSPI peripheral pin |
| IOPORT_PERIPHERAL_SSI | Pin will function as an SSI peripheral pin |
| IOPORT_PERIPHERAL_USB_FS | Pin will function as a USB full speed peripheral pin |
| IOPORT_PERIPHERAL_USB_HS | Pin will function as a USB high speed peripheral pin |
| IOPORT_PERIPHERAL_GPT2 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_SDHI_MMC | Pin will function as an SD/MMC peripheral pin |
| IOPORT_PERIPHERAL_GPT3 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_MII | Pin will function as an Ethernet MMI peripheral pin |
| IOPORT_PERIPHERAL_GPT4 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_GPT5 | Pin will function as a GPT peripheral pin |
| IOPORT_PERIPHERAL_ETHER_RMII | Pin will function as an Ethernet RMMI peripheral pin |
| IOPORT_PERIPHERAL_PDC | Pin will function as a PDC peripheral pin |
| IOPORT_PERIPHERAL_LCD_GRAPHICS | Pin will function as a graphics LCD peripheral pin |

| | |
|---|---|
| IOPORT_PERIPHERAL_CAC | Pin will function as a CAC peripheral pin |
| IOPORT_PERIPHERAL_TRACE | Pin will function as a debug trace peripheral pin |
| IOPORT_PERIPHERAL_OSPI | Pin will function as a OSPI peripheral pin |
| IOPORT_PERIPHERAL_CEC | Pin will function as a CEC peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT0 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_PGAOUT1 | Pin will function as a PGAOUT peripheral pin |
| IOPORT_PERIPHERAL_ULPT | Pin will function as a ULPT peripheral pin |
| IOPORT_PERIPHERAL_MIPI | Pin will function as a MIPI DSI peripheral pin |
| IOPORT_PERIPHERAL_UARTA | Pin will function as an UARTA peripheral pin |

### ◆ ioport_cfg_options_t

| enum ioport_cfg_options_t | |
|---|---|
| Options to configure pin functions | |
| Enumerator | |
| IOPORT_CFG_PORT_DIRECTION_INPUT | Sets the pin direction to input (default) |
| IOPORT_CFG_PORT_DIRECTION_OUTPUT | Sets the pin direction to output. |
| IOPORT_CFG_PORT_OUTPUT_LOW | Sets the pin level to low. |
| IOPORT_CFG_PORT_OUTPUT_HIGH | Sets the pin level to high. |
| IOPORT_CFG_PULLUP_ENABLE | Enables the pin's internal pull-up. |
| IOPORT_CFG_PIM_TTL | Enables the pin's input mode. |
| IOPORT_CFG_NMOS_ENABLE | Enables the pin's NMOS open-drain output. |
| IOPORT_CFG_IRQ_ENABLE | Sets pin as an IRQ pin. |
| IOPORT_CFG_ANALOG_ENABLE | Enables pin to operate as an analog pin. |
| IOPORT_CFG_PERIPHERAL_PIN | Enables pin to operate as a peripheral pin. |
| IOPORT_CFG_PORT_DIRECTION_INPUT | Sets the pin direction to input (default) |

| | |
|---|---|
| IOPORT_CFG_PORT_DIRECTION_OUTPUT | Sets the pin direction to output. |
| IOPORT_CFG_PORT_OUTPUT_LOW | Sets the pin level to low. |
| IOPORT_CFG_PORT_OUTPUT_HIGH | Sets the pin level to high. |
| IOPORT_CFG_PULLUP_ENABLE | Enables the pin's internal pull-up. |
| IOPORT_CFG_PIM_TTL | Enables the pin's input mode. |
| IOPORT_CFG_NMOS_ENABLE | Enables the pin's NMOS open-drain output. |
| IOPORT_CFG_IRQ_ENABLE | Sets pin as an IRQ pin. |
| IOPORT_CFG_ANALOG_ENABLE | Enables pin to operate as an analog pin. |
| IOPORT_CFG_PERIPHERAL_PIN | Enables pin to operate as a peripheral pin. |
| IOPORT_CFG_PORT_DIRECTION_INPUT | Sets the pin direction to input (default) |
| IOPORT_CFG_PORT_DIRECTION_OUTPUT | Sets the pin direction to output. |
| IOPORT_CFG_PORT_OUTPUT_LOW | Sets the pin level to low. |
| IOPORT_CFG_PORT_OUTPUT_HIGH | Sets the pin level to high. |
| IOPORT_CFG_PULLUP_ENABLE | Enables the pin's internal pull-up. |
| IOPORT_CFG_PIM_TTL | Enables the pin's input mode. |
| IOPORT_CFG_NMOS_ENABLE | Enables the pin's NMOS open-drain output. |
| IOPORT_CFG_IRQ_ENABLE | Sets pin as an IRQ pin. |
| IOPORT_CFG_ANALOG_ENABLE | Enables pin to operate as an analog pin. |
| IOPORT_CFG_PERIPHERAL_PIN | Enables pin to operate as a peripheral pin. |
| IOPORT_CFG_PORT_DIRECTION_INPUT | Sets the pin direction to input (default) |
| IOPORT_CFG_PORT_DIRECTION_OUTPUT | Sets the pin direction to output. |
| IOPORT_CFG_PORT_OUTPUT_LOW | Sets the pin level to low. |
| IOPORT_CFG_PORT_OUTPUT_HIGH | Sets the pin level to high. |
| IOPORT_CFG_PULLUP_ENABLE | Enables the pin's internal pull-up. |

| | |
|---|---|
| IOPORT_CFG_PIM_TTL | Enables the pin's input mode. |
| IOPORT_CFG_NMOS_ENABLE | Enables the pin's NMOS open-drain output. |
| IOPORT_CFG_PMOS_ENABLE | Enables the pin's PMOS open-drain output. |
| IOPORT_CFG_DRIVE_MID | Sets pin drive output to medium. |
| IOPORT_CFG_DRIVE_HS_HIGH | Sets pin drive output to high along with supporting high speed. |
| IOPORT_CFG_DRIVE_MID_IIC | Sets pin to drive output needed for IIC on a 20mA port. |
| IOPORT_CFG_DRIVE_HIGH | Sets pin drive output to high. |
| IOPORT_CFG_EVENT_RISING_EDGE | Sets pin event trigger to rising edge. |
| IOPORT_CFG_EVENT_FALLING_EDGE | Sets pin event trigger to falling edge. |
| IOPORT_CFG_EVENT_BOTH_EDGES | Sets pin event trigger to both edges. |
| IOPORT_CFG_IRQ_ENABLE | Sets pin as an IRQ pin. |
| IOPORT_CFG_ANALOG_ENABLE | Enables pin to operate as an analog pin. |
| IOPORT_CFG_PERIPHERAL_PIN | Enables pin to operate as a peripheral pin. |

### ◆ ioport_output_current_t

| enum ioport_output_current_t | |
|---|---|
| Output current settings for CCD pins | |
| Enumerator | |
| IOPORT_OUTPUT_CURRENT_HI_Z | High-impedance state. |
| IOPORT_OUTPUT_CURRENT_2_MA | 2 mA output current |
| IOPORT_OUTPUT_CURRENT_5_MA | 5 mA output current |
| IOPORT_OUTPUT_CURRENT_10_MA | 10 mA output current |
| IOPORT_OUTPUT_CURRENT_15_MA | 15 mA output current |
| IOPORT_OUTPUT_CURRENT_DISABLE | Disables output current control. |

## Function Documentation

### ◆ R_IOPORT_Open()

fsp_err_t R_IOPORT_Open ( ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg* )

Initializes internal driver data, then calls pin configuration function to configure pins.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configuration data written to PFS register(s) |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_ARGUMENT | Pin configurations is invalid. |

### ◆ R_IOPORT_Close()

fsp_err_t R_IOPORT_Close ( ioport_ctrl_t *const *p_ctrl*)

Resets IOPORT registers. Implements ioport_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The IOPORT was successfully uninitialized |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The module has not been opened |

### ◆ R_IOPORT_PinsCfg()

fsp_err_t R_IOPORT_PinsCfg ( ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg* )

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg.

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the Pins tab of the RA Configuration editor or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and testing.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configuration data written to PFS register(s) |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

### ◆ R_IOPORT_PinCfg()

fsp_err_t R_IOPORT_PinCfg ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, uint32_t *cfg* )

Configures the settings of a pin. Implements ioport_api_t::pinCfg.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin configured |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different pins. This function will change the configuration of the pin with the new configuration. For example it is not possible with this function to change the drive strength of a pin while leaving all the other pin settings unchanged. To achieve this the original settings with the required change will need to be written using this function.*

### ◆ R_IOPORT_PinRead()

fsp_err_t R_IOPORT_PinRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_value* )

Reads the level on a pin. Implements ioport_api_t::pinRead.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |

***Note***

*This function is re-entrant for different pins.*

### ◆ R_IOPORT_PortRead()

fsp_err_t R_IOPORT_PortRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_port_value* )

Reads the value on an IO port. Implements ioport_api_t::portRead.

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortWrite()

fsp_err_t R_IOPORT_PortWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *value*, ioport_size_t *mask* )

Writes to multiple pins on a port. Implements ioport_api_t::portWrite.

The input value will be written to the specified port. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Only the bits with the corresponding bit in the mask value set will be updated. For example, value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port written to |
| FSP_ERR_INVALID_ARGUMENT | The port and/or mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different ports. This function makes use of the PCNTR3 register to atomically modify the levels on the specified pins on a port.*

### ◆ R_IOPORT_PinWrite()

fsp_err_t R_IOPORT_PinWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *level* )

Sets a pin's output either high or low. Implements ioport_api_t::pinWrite.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin written to |
| FSP_ERR_INVALID_ARGUMENT | The pin and/or level not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different pins. This function makes use of the PCNTR3 register to atomically modify the level on the specified pin on a port.*

### ◆ R_IOPORT_PortDirectionSet()

fsp_err_t R_IOPORT_PortDirectionSet ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *direction_values*, ioport_size_t *mask* )

Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet().

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. If a bit is set to 1 then the corresponding pin will be changed to an input or an output as specified by the direction values. If a mask bit is set to 0 then the direction of the pin will not be changed.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port direction updated |
| FSP_ERR_INVALID_ARGUMENT | The port and/or mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PortEventInputRead()

fsp_err_t R_IOPORT_PortEventInputRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_event_data* )

Reads the value of the event input data. Implements ioport_api_t::portEventInputRead().

The event input data for the port will be read. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

The port event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port read |
| FSP_ERR_INVALID_ARGUMENT | Port not a valid ELC port |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_UNSUPPORTED | Function not supported. |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PinEventInputRead()

fsp_err_t R_IOPORT_PinEventInputRead ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_event* )

Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead.

The pin event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin read |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_INVALID_ARGUMENT | Port is not valid ELC PORT. |
| FSP_ERR_UNSUPPORTED | Function not supported. |

***Note***

*This function is re-entrant.*

### ◆ R_IOPORT_PortEventOutputWrite()

fsp_err_t R_IOPORT_PortEventOutputWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *event_data*, ioport_size_t *mask_value* )

This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite.

Using the event system enables a port state to be stored by this function in advance of being output on the port. The output to the port will occur when the ELC event occurs.

The input value will be written to the specified port when an ELC event configured for that port occurs. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Port event data written |
| FSP_ERR_INVALID_ARGUMENT | Port or Mask not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_UNSUPPORTED | Function not supported |

***Note***

*This function is re-entrant for different ports.*

### ◆ R_IOPORT_PinEventOutputWrite()

fsp_err_t R_IOPORT_PinEventOutputWrite ( ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *pin_value* )

This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite.

Using the event system enables a pin state to be stored by this function in advance of being output on the pin. The output to the pin will occur when the ELC event occurs.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Pin event data written |
| FSP_ERR_INVALID_ARGUMENT | Port or Pin or value not valid |
| FSP_ERR_NOT_OPEN | The module has not been opened |
| FSP_ERR_ASSERTION | NULL pointer |
| FSP_ERR_UNSUPPORTED | Function not supported |

***Note***

*This function is re-entrant for different ports.*

### 5.2.18.4 Inter-Processor Communication (r_ipc)

Modules » System

**Functions**

| | |
|---|---|
| fsp_err_t | R_IPC_Open (ipc_ctrl_t *const p_api_ctrl, ipc_cfg_t const *const p_cfg) |
| fsp_err_t | R_IPC_MessageSend (ipc_ctrl_t *const p_api_ctrl, uint32_t message) |
| fsp_err_t | R_IPC_EventGenerate (ipc_ctrl_t *const p_api_ctrl, ipc_generate_event_t event) |
| fsp_err_t | R_IPC_CallbackSet (ipc_ctrl_t *const p_api_ctrl, void(*p_callback)(ipc_callback_args_t *), void *const p_context, ipc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_IPC_Close (ipc_ctrl_t *const p_api_ctrl) |

**Detailed Description**

Driver for the IPC peripheral on RA MCUs. This module implements the IPC Interface.

# Overview

The IPC HAL module supports communication across processors via maskable interrupts and one-way message FIFOs.

*Note*

*IPC semaphores and NMI are handled through BSP APIs. See Inter-Processor Communication (IPC) Support.*

**Features**

- Send and receive messages through IPC FIFOs
- Trigger or receive maskable interrupts with multiple event bits settable per interrupt

# Configuration

**Build Time Configurations for r_ipc**

The following build time configurations are defined in fsp_cfg/r_ipc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

**Configurations for System > IPC (r_ipc)**

This module can be added to the Stacks tab via New Stack > System > IPC (r_ipc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_ipc0 | Module name. |
| Channel | MCU Specific Options | | Select the IPC channel. |
| Callback | Name must be a valid C symbol | NULL | A user callback function must be provided when the IPC interrupt is enabled. It will be called from the interrupt service routine (ISR). |
| Interrupt Priority | MCU Specific Options | | Select the IPC interrupt priority. An interrupt is only required for receiving messages and events from the opposing core. |

### Interrupt Configuration

- In order to receive messages, the interrupt must be enabled and a callback must be supplied. If an IPC instance is only being used to send messages or generate events the interrupt doesn't have to be enabled.

# Usage Notes

IPC maskable interrupts and FIFOs share the same interrupt so the IPC driver handles configuration of both similarly:

- The IPC driver will choose which IPC registers to use based on the selected core and channel.
- An interrupt and callback is only required when the IPC instance is being used to receive.
    - When receiving a callback should be used to process any received messages/interrupts.
- The channels across cores should always match up for IPC instances. For example, in order to communicate from Core 0 to Core 1 over channel 0:
    - Core 0 should have an IPC instance configured opened on channel 0
    - Core 1 should have an IPC instance configured opened on channel 0

### IPC Maskable Interrupt Events

Each IPC channel can set multiple "event" bits in a single maskable interrupt. This should be done through ipc_api_t::eventGenerate.

# Examples

### Basic IPC Messaging Example

The following example demonstrates sending a simple message from Core 0 to Core 1 via IPC.

### Core 0 Application

```
#define IPC_SEND_MESSAGE 0xABCD
void r_ipc_basic_messaging_core0_example (void)
{
 /* Open an IPC instance on the core 0 application configured to send a message to
core 1.
  * @ref ipc_cfg_t::direction should be set to IPC_DIRECTION_SEND.
  * @ref ipc_cfg_t::channel should match for IPC instances on both cores.
  */
 fsp_err_t err = R_IPC_Open(&g_ipc0_ctrl, &g_ipc0_cfg);
    assert(FSP_SUCCESS == err);
 /* Send IPC_SEND_MESSAGE to core 1. */
    err = R_IPC_MessageSend(&g_ipc0_ctrl, IPC_SEND_MESSAGE);
    assert(FSP_SUCCESS == err);
}
```

### Core 1 Application

```
void r_ipc_basic_messaging_core1_example (void)
{
 /* Open an IPC instance on the core 1 application configured to receive messages.
  * @ref ipc_cfg_t::direction should be set to IPC_DIRECTION_RECEIVE.
  * @ref ipc_cfg_t::channel should match for IPC instances on both cores.
  */
 fsp_err_t err = R_IPC_Open(&g_ipc0_ctrl, &g_ipc0_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for message from callback */
 while (!g_callback_called)
    {
       ;
    }
 /* Message will be in g_last_message_received */
    (void) g_last_message_received;
```

```
}
void ipc_message_example_callback (ipc_callback_args_t * p_args)
{
 /* Check for message received event */
 if (IPC_EVENT_MESSAGE_RECEIVED & p_args->event)
    {
       g_callback_called       = true;
       g_last_message_received = p_args->message;
    }
}
```

### Basic IPC Interrupt Event Example

The following example demonstrates triggering an event via maskable interrupt from Core 0 to Core 1.

### Core 0 Application

```
void r_ipc_basic_interrupt_event_core0_example (void)
{
 /* Open an IPC instance on the core 0 application configured to send to core 1.
  * @ref ipc_cfg_t::direction should be set to IPC_DIRECTION_SEND.
  * @ref ipc_cfg_t::channel should match for IPC instances on both cores.
  */
 fsp_err_t err = R_IPC_Open(&g_ipc0_ctrl, &g_ipc0_cfg);
    assert(FSP_SUCCESS == err);
 /* Trigger interrupt on core 1 with event 0 bit set. */
    err = R_IPC_EventGenerate(&g_ipc0_ctrl, IPC_GENERATE_EVENT_IRQ0);
    assert(FSP_SUCCESS == err);
}
```

### Core 1 Application

```
void r_ipc_basic_interrupt_event_core1_example (void)
{
 /* Open an IPC instance on the core 1 application configured to receive.
  * @ref ipc_cfg_t::direction should be set to IPC_DIRECTION_RECEIVE.
```

```
 * @ref ipc_cfg_t::channel should match for IPC instances on both cores.
 */
fsp_err_t err = R_IPC_Open(&g_ipc0_ctrl, &g_ipc0_cfg);
   assert(FSP_SUCCESS == err);
/* Wait for event */
while (!g_event0_triggered)
   {
      ;
   }
}
void ipc_event_example_callback (ipc_callback_args_t * p_args)
{
/* Check for message received event */
if (IPC_EVENT_IRQ0 & p_args->event)
   {
      g_event0_triggered = true;
   }
}
```

### Function Documentation

### ◆ R_IPC_Open()

fsp_err_t R_IPC_Open ( ipc_ctrl_t *const *p_api_ctrl*, ipc_cfg_t const *const *p_cfg* )

Configures the IPC driver channel based on the input configuration.

Implements ipc_api_t::open.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel opened successfully. |
| FSP_ERR_ASSERTION | Pointer to IPC control block or configuration structure is NULL. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The requested channel does not exist on this MCU. |
| FSP_ERR_ALREADY_OPEN | Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_IPC_MessageSend()

fsp_err_t R_IPC_MessageSend ( ipc_ctrl_t *const *p_api_ctrl*, uint32_t *message* )

Sends a message over the configured IPC channel.

Implements ipc_api_t::messageSend.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Message written to FIFO successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_INSUFFICIENT_SPACE | IPC channel FIFO is full |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_IPC_EventGenerate()

fsp_err_t R_IPC_EventGenerate ( ipc_ctrl_t *const *p_api_ctrl*, ipc_generate_event_t *event* )

Triggers an interrupt with the specified event bit set over the configured IPC channel.

Implements ipc_api_t::eventGenerate.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Event request set successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_IPC_CallbackSet()

fsp_err_t R_IPC_CallbackSet ( ipc_ctrl_t *const *p_api_ctrl*, void(*)(ipc_callback_args_t *) *p_callback*, void *const *p_context*, ipc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure.

Implements ipc_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | Pointer to IPC control block or callback is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

### ◆ R_IPC_Close()

fsp_err_t R_IPC_Close ( ipc_ctrl_t *const *p_api_ctrl*)

Closes the IPC driver.

Implements ipc_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Channel successfully closed. |
| FSP_ERR_ASSERTION | Pointer to IPC control block is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

**Returns**

See Common Error Codes or functions called by this function for other possible return codes.

## 5.2.19 Timers

Modules

### Detailed Description

Timers Modules.

### Modules

| | |
|---|---|
| | Independent Channel, 16-bit and 8-bit timer (r_tau)<br>Driver for the TAU peripheral on RA MCUs. This module implements the Timer Interface. |
| | Port Output Enable for GPT (r_poeg)<br>Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface. |
| | Realtime Clock (r_rtc)<br>Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface. |
| | Realtime Clock (r_rtc_c)<br>Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface. |

| | |
|---|---|
| | Three-Phase PWM (r_gpt_three_phase)<br>Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface. |
| | Timer, 32-bit Interval Timer (r_tml)<br>Driver for the TML peripherals on RA MCUs. This module implements the Timer Interface. |
| | Timer, General PWM (r_gpt)<br>Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface. |
| | Timer, Low-Power (r_agt)<br>Driver for the AGT and AGTW peripheral on RA MCUs. This module implements the Timer Interface. |
| | Timer, Simultaneous Channel (r_tau_pwm)<br>Driver for the TAU_PWM peripheral on RA MCUs. This module implements the Timer Interface. |
| | Timer, Ultra Low-Power (r_ulpt)<br>Driver for the ULPT peripheral on RA MCUs. This module implements the Timer Interface. |

### 5.2.19.1 Independent Channel, 16-bit and 8-bit timer (r_tau)

Modules » Timers

**Functions**

| | |
|---|---|
| fsp_err_t | R_TAU_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_TAU_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_Reset (timer_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t | R_TAU_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_TAU_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_TAU_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_TAU_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_TAU_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_TAU_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_TAU_Close (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_LinMeasurementFuncSwitch (timer_ctrl_t *const p_ctrl, tau_function_t func) |

### Detailed Description

Driver for the TAU peripheral on RA MCUs. This module implements the Timer Interface.

# Overview

### Features

The TAU module has the following features:

- Supports Function Interval Timer, Square Wave Output, External Event Counter, Divider Function, Input Pulse Interval Measurement, Delay Counter, Input Signal High- or Low-Level Width Measurement.
- Configurable clock source and external sources input to TImn
- Configurable period (counts per timer cycle).
- Supports operation clocks: CK00, CK01, CK02, and CK03.
- Supports noise filter on input source.
- Supports 16-bit and 8-bit timers.
- Supports the runtime reconfiguration of the period.
- Signal can be output to a pin.
- Supports counting based on an external signal to TImn
- Supports measuring pulse width or pulse period.
- APIs are provided to start, stop, and reset the counter.

- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value. Below table illustrates the channels and their corresponding supported features.

| Channel | Interval Timer | Square Wave Output | External Event Counter | Divider | Input Pulse Interval Measurement | Delay Counter | Measure High-Low Level Pulse Width | 8-bit timer operation | Operation clocks | Input signals | Output signals |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TAU00 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | CK00, CK01 | TI00, ELC | TO00 |
| TAU01 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | CK00, CK01, CK02, CK03 | TI01, ELC | TO01 |
| TAU02 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | CK00, CK01 | TI02 | TO02 |
| TAU03 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | CK00, CK01, CK02, CK03 | TI03 | TO03 |
| TAU04 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | CK00, CK01 | TI04 | TO04 |
| TAU05 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | CK00, CK01 | TI05, MOCO, LOCO, FSUB | TO05 |
| TAU06 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | CK00, CK01 | TI06 | TO06 |
| TAU07 | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | CK00, CK01 | TI07, RXD2 (*) | TO07 |

(*) Remark: RXD0 for the MCU with only SAU0.

# Configuration

### Build Time Configurations for r_tau

The following build time configurations are defined in fsp_cfg/r_tau_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled | Default (BSP) | If selected code for parameter checking is |

| | | | |
|---|---|---|---|
| | • Disabled | | included in the build. |
| Interrupt Support | • Disabled<br>• Enabled | Enabled | Enable support for interrupts |
| Pin Output Support | • Disabled<br>• Enabled | Disabled | Enable output for either square wave or divider output |
| Pin Input Support | • Disabled<br>• Enabled | Disabled | Enable input for pulse width measurement, level width measurement, pulse/edge counting or divider functions. |
| Extra Input Mode Support | • Disabled<br>• Enabled | Disabled | Enable support for event counting, system clock count sources and input noise filtering when Pin Input Support is enabled. |
| 8-Bit Mode Support | • Disabled<br>• Enabled | Disabled | Enable support for 8-bit timer modes (only available on channels 1 and 3). |

**Configurations for Timers > Timer, Independent Channel, 16-bit and 8-bit Timer Operation (r_tau)**

This module can be added to the Stacks tab via New Stack > Timers > Timer, Independent Channel, 16-bit and 8-bit Timer Operation (r_tau).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Channel | Channel number must be a non-negative integer | 0 | Physical hardware channel. |
| Function | • Interval Timer<br>• Square Wave Output<br>• External Event Counter<br>• Divider<br>• Input Pulse Interval Measurement<br>• Measure Low Level Pulse Width<br>• Measure High | Interval Timer | Function selection. Note: The calculation of the input pulse function and the high-low level pulse width measurement function is implemented using interrupts. ISR's must be enabled for these functions even if callback is unused. |

| | | | |
|---|---|---|---|
| | Level Pulse Width<br>• Delay Counter | | |
| Bit Timer Mode | • 16-bit timer<br>• Higher 8-bit timer<br>• Lower 8-bit timer<br>• Higher and Lower 8-bit timer | 16-bit timer | Specify the 16 or 8-bit timer mode |
| Operation Clock | • CK00<br>• CK01<br>• CK02<br>• CK03 | CK00 | Specify the selection of operation clock |
| Period | Value must be a non-negative integer | 0x10000 | Specify the timer period based on the selected units.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x10000 results in the maximum period at the lowest divisor (fastest timer tick). When Pulse width measurement or high-low level pulse width measurement is enabled, this period is always generated as zero.. The period should be set maximally 0x100 when lower 8-bit mode or higher and lower 8-bit mode is enabled. When 'Input->Trigger Edge' is set to 'Trigger Edge Rising' or 'Trigger Edge Falling', this period can be maximally set to 0x20000 when using the divider function.<br><br>If the requested period cannot be achieved, the settings with the largest possible period, that does not exceed the requested period, are used. The |

| | | | |
|---|---|---|---|
| | | | theoretical calculated period is printed in a comment in the timer_cfg_t structure. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Period (Higher 8-bit timer) | Value must be a non-negative integer | 0x100 | Specify the higher 8-bit timer period based on the selected unit.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x100 results in the maximum period at the lowest divisor (fastest timer tick). This setting is only applicable in interval timer function with higher 8bit mode or higher and lower 8bit mode.<br><br>If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the tau_extended_cfg_t structure. |
| Period Unit (Higher 8-bit timer) | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Input | | | |
| Input Source | MCU Specific Options | | Input source, applies in input pulse width measurement function, high-low level pulse |

| | | | |
|---|---|---|---|
| | | | width measurement function, external event count function, and divider function. |
| Trigger Edge | • Trigger Edge Rising<br>• Trigger Edge Falling<br>• Trigger Edge Both | Trigger Edge Rising | Select the trigger edge. Applies in input pulse width measurement function, high-low level pulse width measurement function, external event count function, and divider function. |
| Input Filter | • Disabled<br>• Enabled | Disabled | Input filter, applies in input pulse width measurement function, high-low level pulse width measurement function, external event count function, and divider function. |
| Output | | | |
| Initial Output | • Disabled<br>• Start Level High<br>• Start Level Low | Disabled | Initial output, applies in divider function and square wave. |
| Interrupts | | | |
| Setting of starting count and interrupt | • Timer interrupt is not generated when counting is started/Start trigger is invalid during counting operation.<br>• Timer interrupt is generated when counting is started/Start trigger is valid during counting operation. | Timer interrupt is not generated when counting is started/Start trigger is invalid during counting operation. | Specify OPIRQ bit setting |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses. |
| Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

| | | |
|---|---|---|
| Higher 8-bit Interrupt Priority | MCU Specific Options | Timer higher 8-bit interrupt priority. |

### Clock Configuration

The TAU clock is based on the peripheral module clock (PCLKB) which is equal to the system clock (ICLK).

Each TAU channel has certain operation clocks selections, these can be set with the **General>Operation Clock** property in the module configuration. When the operation clock of a channel is set to CK00, CK01, CK02, or CK03, the TAU module provides divisor values for each of those clocks. These divisors may be set in the **Clocks** tab. As such, setting a divisor in the **Clocks** tab affects all TAU channels that use that CK0x clock as an input. Adjusting these settings determines the frequency range achievable by a TAU channel. If a desired frequency is not achievable, the divider in the **Clocks** tab may be adjusted. The clock dividers cannot be adjusted at runtime.

### Pin Configuration

This module should use the TOmn pins as output pins for Square Wave Output, Divider Function

For input source, the input signal must be applied to External Event Counter, Divider, Input Pulse Interval Measurement, Delay Counter, and Measure High-Low Level Pulse Width Function

### Timer Period

The RA Configuration editor will automatically calculate the period count value based on the selected period time, units, operation clock (set by **General>Operation Clock** property in the module configuration), and clock divisor value (set in **Clocks** tab).

When the selected unit is "Raw counts", the maximum and minimum allowed period setting varies depending on the selected clock source:

| Clock divisor | Minimum period (counts) | Maximum period (counts) |
|---|---|---|
| ICLK/1 | 0x00002 | 0x10000 |
| All other divisors | 0x00001 | 0x10000 |

*Note*

*If ICLK (undivided) is selected as the operation clock (CK00, CK01) and TDR0n is set to 0x0000 (n = 0 to 7), interrupt requests output from timer array units cannot be used. For this reason When Clock Source is ICLK, Minimum period (counts) is 0x00002*

| Clock divisor | Minimum Higher 8-bit period (counts) | Maximum Higher 8-bit period (counts) |
|---|---|---|
| All divisors | 0x001 | 0x100 |

*Note*

*Though the TAU is a 16-bit timer, because the period interrupt occurs when the counter underflow, setting the period register to 0 results in an effective period of 1 count. For this reason all user-provided raw count values reflect the actual number of period counts (not the raw register values).*

# Usage Notes

### Interval Timer Function

This function can be used as a reference timer to generate TAU0_TMI0n (timer interrupt) at fixed intervals. The interrupt generation period can be calculated by the following expression.

- Generation period of TAU0_TMI0n (timer interrupt) = Period of count clock × (Set value of TDR0n + 1)

The Interval Timer Function can operate as the higher/lower 8-bits mode.

When "Interrupts > Setting of starting count and interrupt" is "Timer interrupt is generated when counting is started/Start trigger is valid during counting operation.", an interrupt is generated immediately after R_TAU_Start() function is called and timer output also changed.

### Square Wave Output Function

This function can use TO0n to perform a toggle operation as soon as TAU0_TMI0n has been generated, and outputs a square wave with a duty factor of 50%.

The period and frequency for outputting a square wave from TO0n can be calculated by the following expressions.

- Period of square wave output from TO0n = Period of count clock × (Set value of TDR0n + 1) × 2
- Frequency of square wave output from TO0n = Frequency of count clock / {(Set value of TDR0n + 1) × 2}

The Square Wave Output Function can operate as the lower 8-bits mode.

When "Interrupts > Setting of starting count and interrupt" is "Timer interrupt is generated when counting is started/Start trigger is valid during counting operation.", an interrupt is generated immediately after R_TAU_Start() function is called and output pin is also changed.

### External Event Counter Function

The timer array unit can be used as an external event counter that counts the number of times the valid input edge (external event) is detected in the TI0n pin. When a specified count value is reached, the event counter generates an interrupt. The specified number of counts can be calculated by the following expression.

- Specified number of counts = Set value of TDR0n + 1

Instead of using the TI0n pin input, a channel specified for the External Event Counter Function can also use the signal as "Middle-speed on-chip oscillator (MOCO)", "Low-speed on-chip oscillator (LOCO)", "Subsystem clock (FSUB)", and ELC event which selected in **Input>Input Source** property as its input source to drive counting.

The External Event Counter Function can operate as the lower 8-bits mode.

### Delay Counter Function

It is possible to start counting down when the valid edge of the TI0n pin input is detected (an

external event), and then generate TAU0_TMI0n (a timer interrupt) after any specified interval. The interrupt generation period can be calculated by the following expression. Generation period of TAU0_TMI0n (timer interrupt) = Period of count clock × (Set value of TDR0n + 1)

Instead of using the TI0n pin input, a channel specified for the Delay Counter Function can also use the signal as "Middle-speed on-chip oscillator (MOCO)", "Low-speed on-chip oscillator (LOCO)", "Subsystem clock (FSUB)", and ELC event which selected in **Input>Input Source** property as its input source to drive counting.

The Delay Counter Function can operate as the lower 8-bits mode.

When "Interrupts > Setting of starting count and interrupt" is "Timer interrupt is generated when counting is started/Start trigger is valid during counting operation.", a start trigger is valid during counting operation.

**Divider Function**

The timer array unit can be used as a frequency divider that divides a clock input to the TI00 pin and outputs the result from the TO00 pin. The divided clock frequency output from TO00 can be calculated by the following expression. When rising edge/falling edge is selected:

- Divided clock frequency = Input clock frequency / {(Set value of TDR00 + 1) × 2} When both edges are selected:
- Divided clock frequency ≈ Input clock frequency / (Set value of TDR00 + 1)

When "Interrupts > Setting of starting count and interrupt" is "Timer interrupt is generated when counting is started/Start trigger is valid during counting operation.", an interrupt is generated immediately after first input signal is detected and output pin is also changed.

**Input Pulse Interval Measurement Function**

The count value can be captured at the TI0n valid edge and the interval of the pulse input to TI0n can be measured. The pulse interval can be calculated by the following expression.

- TI0n input pulse interval = Period of count clock × ((0x10000 × TSR0n.OVF) + (Captured value of TDR0n + 1))

Instead of using the TI0n pin input, a channel specified for the Input Pulse Interval Measurement can also use the signal as "Middle-speed on-chip oscillator (MOCO)", "Low-speed on-chip oscillator (LOCO)", "Subsystem clock (FSUB)", "Input signal of the RXD pin", and ELC event which selected in **Input>Input Source** property as its input source to drive counting.

**High-Low Level Pulse Width Measurement**

By starting counting at one edge of the TI0n pin input and capturing the number of counts at another edge, the signal width (high-level width or low-level width) of TI0n can be measured. The signal width of TI0n can be calculated by the following expression.

- Signal width of TI0n input = Period of count clock × ((0x10000 × TSR0n.OVF) + (Captured value of TDR0n + 1))

Instead of using the TI0n pin input, a channel specified for the High-Low Level Pulse Width Function can also use the signal as "Middle-speed on-chip oscillator (MOCO)", "Low-speed on-chip oscillator (LOCO)", "Subsystem clock (FSUB)", "Input signal of the RXD pin", and ELC event which selected in **Input>Input Source** property as its input source to drive counting.

### Triggering ELC Events with TAU

The TAU timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

### Limitations

- When using pin input a delay of 2 TAU input clocks (ICLK/CK0*) is required between R_TAU_Open() and R_TAU_Start(). When using the noise filter, a delay of 4 clocks is required instead. See in hardware manual: TAU > Timer Input (TI0n) Control > Cautions on Channel Input Operation.
- When using High-Low Level Pulse Width Measurement function, the counter can not be reset by invoking R_TAU_Reset().

# Examples

### TAU Basic Example

This is a basic example of minimal use of the TAU in an application.

```
void tau_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_Start(&g_timer0_ctrl);
}
```

### TAU Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
uint32_t g_counter_underflow = 0U;
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
```

```
 /* Add application code to be called periodically here. */
       g_counter_underflow++;
    }
 if (TIMER_EVENT_HIGHER_8BIT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### TAU 16-bit or 8-bit mode Example

To use the TAU as 16-bit or 8-bit mode, select Bit Mode Timer.

```
void tau_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_TAU_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_StatusGet(&g_timer0_ctrl, &status);
}
```

```
void tau_higher_8bit_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_Open(&g_timer0_ctrl, &g_timer0_higher_8bit_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_TAU_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_StatusGet(&g_timer0_ctrl, &status);
}
```

### TAU Input Capture Example

This is an example of using the TAU to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint32_t g_captured_time = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CAPTURE_EDGE == p_args->event)
    {
 /* Process capture from TAUIO. */
       g_captured_time = p_args->capture;
    }
}
void tau_input_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_Open(&g_timer_input_capture_ctrl, &g_timer_input_capture_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_TAU_Enable(&g_timer_input_capture_ctrl);
 /* (Optional) Disable captures. */
    (void) R_TAU_Disable(&g_timer_input_capture_ctrl);
```

```
}
```

### TAU Period Update Example

This is an example of updating the period.

```
#define TAU_EXAMPLE_MSEC_PER_SEC (1000)
#define TAU_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void tau_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_Start(&g_timer0_ctrl);
 /* Get operation clock frequency
  * - Use the R_TAU_InfoGet function (it accounts for the clock source and divider of
operation clock ).
  */
 timer_info_t info;
    (void) R_TAU_InfoGet(&g_timer0_ctrl, &info);
    uint32_t timer_freq_hz = info.clock_frequency;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / iclk_freq_hz. A cast to uint32_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) ((timer_freq_hz * TAU_EXAMPLE_DESIRED_PERIOD_MSEC) /
TAU_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
```

```
    err = R_TAU_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

### TAU Square Wave Output Function Example

This is an example of configuration for Square Wave Output Function.

```
/* This example shows how configure squarewave output function. */
void tau_squarewave_output_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers. */
    err = R_TAU_Open(&g_timer_squarewave_ctrl, &g_timer_squarewave_cfg);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_Start(&g_timer_squarewave_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_StatusGet(&g_timer_squarewave_ctrl, &status);
}
```

### TAU External Event Counter Function Example

This is an example of configuration for External Event Counter Function.

```
/* This example shows how to configure the external event counter function. */
void tau_external_event_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers. */
    err = R_TAU_Open(&g_timer_external_event_ctrl, &g_timer_external_event_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable external event counter */
    (void) R_TAU_Enable(&g_timer_external_event_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
```

```
 timer_status_t status;
    (void) R_TAU_StatusGet(&g_timer_external_event_ctrl, &status);
}
```

### TAU Divider Function Example

This is an example of configuration for Divider Function.

```
/* This example shows how configure divider function. */
void tau_divider_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers. */
    err = R_TAU_Open(&g_timer_divider_ctrl, &g_timer_divider_cfg);
    assert(FSP_SUCCESS == err);
 /* Start the divider. */
    (void) R_TAU_Start(&g_timer_divider_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_StatusGet(&g_timer_divider_ctrl, &status);
}
```

### TAU Delay Counter Function Example

This is an example of configuration for Delay Counter Function.

```
/* This example shows how configure delay counter function. */
void tau_delay_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers. */
    err = R_TAU_Open(&g_timer_delay_counter_ctrl, &g_timer_delay_counter_cfg);
    assert(FSP_SUCCESS == err);
 /* Start Delay Counter. */
    (void) R_TAU_Enable(&g_timer_delay_counter_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
```

```
timer_status_t status;
   (void) R_TAU_StatusGet(&g_timer_delay_counter_ctrl, &status);
}
```

### Data Structures

| | |
|---|---|
| struct | tau_instance_ctrl_t |
| struct | tau_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | tau_function_t |
| enum | tau_bit_mode_t |
| enum | tau_operation_ck_t |
| enum | tau_trigger_edge_t |
| enum | tau_interrupt_opirq_bit_t |
| enum | tau_input_source_t |
| enum | tau_pin_output_cfg_t |
| enum | tau_input_noise_filter_t |

## Data Structure Documentation

### ◆ tau_instance_ctrl_t

| struct tau_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

### ◆ tau_extended_cfg_t

| struct tau_extended_cfg_t | | |
|---|---|---|
| Optional TAU extension data structure. | | |
| Data Fields | | |
| tau_interrupt_opirq_bit_t | opirq: 1 | Setting of starting count and interrupt. |
| tau_function_t | tau_func | Setting of Function for TAU. |
| tau_bit_mode_t | bit_mode | Setting of 16-bit timer or 8-bit timer. |
| tau_pin_output_cfg_t | initial_output | Setting of Function for TAU. |

| | | |
|---|---|---|
| tau_input_source_t | input_source | |
| tau_input_noise_filter_t | tau_filter | Input filter for TAU. |
| tau_trigger_edge_t | trigger_edge | Trigger edge to start pulse period measurement or count external event. |
| tau_operation_ck_t | operation_clock | |
| uint16_t | period_higher_8bit_counts | Period in raw higher 8 bit timer counts. |
| uint8_t | higher_8bit_cycle_end_ipl | Cycle higher 8-bit end interrupt priority. |
| IRQn_Type | higher_8bit_cycle_end_irq | Cycle higher 8-bit end interrupt. |

## Enumeration Type Documentation

### ◆ tau_function_t

| enum tau_function_t | |
|---|---|
| Timer function | |
| Enumerator | |
| TAU_FUNCTION_INTERVAL | Interval Timer Function. |
| TAU_FUNCTION_SQUARE_WAVE | Square Wave Function. |
| TAU_FUNCTION_EXTERNAL_EVENT_COUNT | External Event Count Function. |
| TAU_FUNCTION_DIVIDER | Divider Function. |
| TAU_FUNCTION_INPUT_PULSE_INTERVAL_MEASUREMENT | Input Pulse Interval Function. |
| TAU_FUNCTION_LOW_LEVEL_WIDTH_MEASUREMENT | Low Level Width Measure Function. |
| TAU_FUNCTION_HIGH_LEVEL_WIDTH_MEASUREMENT | High Level Width Measure Function. |
| TAU_FUNCTION_DELAY_COUNT | Delay Count Function. |

### ◆ tau_bit_mode_t

| enum tau_bit_mode_t | |
|---|---|
| Timer bit mode | |
| Enumerator | |
| TAU_BIT_MODE_16BIT | 16-Bit Timer Mode |
| TAU_BIT_MODE_HIGHER_8BIT | Higher 8-bit Timer Mode. |
| TAU_BIT_MODE_LOWER_8BIT | Lower 8-bit Timer Mode. |
| TAU_BIT_MODE_HIGHER_LOWER_8BIT | Lower and Higher 8-bit Timer Mode. |

### ◆ tau_operation_ck_t

| enum tau_operation_ck_t |
|---|
| Timer operation clock. |

### ◆ tau_trigger_edge_t

| enum tau_trigger_edge_t | |
|---|---|
| Trigger edge for pulse period measurement mode, high-low measurement, divider, delay counter and event counting mode. | |
| Enumerator | |
| TAU_TRIGGER_EDGE_FALLING | Measurement starts or events are counted on falling edge. |
| TAU_TRIGGER_EDGE_RISING | Measurement starts or events are counted on rising edge. |
| TAU_TRIGGER_EDGE_BOTH | Events are counted on both edges (pulse period mode and high-low measurement) |

### ◆ tau_interrupt_opirq_bit_t

| enum tau_interrupt_opirq_bit_t |
|---|
| Interrupt and starting Count Mode |

### ◆ tau_input_source_t

| enum tau_input_source_t | |
|---|---|
| Input source | |
| Enumerator | |
| TAU_INPUT_SOURCE_TI_PIN | Timer Input Source is input pin. |
| TAU_INPUT_SOURCE_ELC | ELC Timer Input Source. |
| TAU_INPUT_SOURCE_RXD_PIN | Timer Input Source is RXD pin. |
| TAU_INPUT_SOURCE_RXD2_PIN | DEPRECATED, do not use. |
| TAU_INPUT_SOURCE_MOCO | Timer Input Source is MOCO. |
| TAU_INPUT_SOURCE_LOCO | Timer Input Source is LOCO. |
| TAU_INPUT_SOURCE_FSUB | Timer Input Source is FSUB. |
| TAU_INPUT_SOURCE_NONE | No Timer Input Source. |

### ◆ tau_pin_output_cfg_t

| enum tau_pin_output_cfg_t | |
|---|---|
| Level of TAU pin | |
| Enumerator | |
| TAU_PIN_OUTPUT_CFG_START_LEVEL_LOW | Pin level low. |
| TAU_PIN_OUTPUT_CFG_START_LEVEL_HIGH | Pin level high. |
| TAU_PIN_OUTPUT_CFG_DISABLED | Not used as output pin. |

### ◆ tau_input_noise_filter_t

| enum tau_input_noise_filter_t | |
|---|---|
| Input filter, applies TAU in high-low measurement, pulse width measurement, delay counter, divider, or event counter mode. | |
| Enumerator | |
| TAU_INPUT_NOISE_FILTER_DISABLE | Disable noise filter. |
| TAU_INPUT_NOISE_FILTER_ENABLE | Enable noise filter. |

## Function Documentation

### ◆ R_TAU_Open()

fsp_err_t R_TAU_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the timer module and applies configurations. Implements timer_api_t::open.

The TAU implementation of the general timer can accept a tau_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_TAU_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the period is not in the valid range |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_CHANNEL | The channel does not support this feature. |
| FSP_ERR_IRQ_BSP_DISABLED | Timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback |
| FSP_ERR_INVALID_MODE | Invalid configuration for p_extend. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |
| FSP_ERR_UNSUPPORTED | Input source is not supported. |

### ◆ R_TAU_Stop()

fsp_err_t R_TAU_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_TAU_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_Start()

fsp_err_t R_TAU_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_TAU_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_Reset()

fsp_err_t R_TAU_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the period minus one. Input Pulse Function is reset counter value to 0. Implements timer_api_t::reset.

***Note***

*This function can not reset the counter when counter is stopped, and function High/Low Measurement Function is used.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The counter value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_Enable()

fsp_err_t R_TAU_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_TAU_Enable(&g_timer_input_capture_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_Disable()

fsp_err_t R_TAU_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

***Note***

*The timer could be running after R_TAU_Disable(). To ensure it is stopped, call R_TAU_Stop().*

Example:

```
/* (Optional) Disable captures. */
   (void) R_TAU_Disable(&g_timer_input_capture_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PeriodSet()

fsp_err_t R_TAU_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Sets period value provided. If the timer is running, the period will be updated after the next counter underflow. If the timer is stopped, this function resets the counter and updates the period. This Function is not supported for Input pulse Function, High-Low Measurement Function. Implements timer_api_t::periodSet.

***Note***

*if timer mode is Lower and Higher 8-bit Timer Mode, the last 8 bits are the lower 8-bits of the timer, and the subsequent 8 bits are the higher 8-bits of the timer*

Example:

```
/* Get operation clock frequency
 * - Use the R_TAU_InfoGet function (it accounts for the clock source and divider of
operation clock ).
 */
timer_info_t info;
   (void) R_TAU_InfoGet(&g_timer0_ctrl, &info);
   uint32_t timer_freq_hz = info.clock_frequency;
/* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
 * desired period is larger than UINT32_MAX / iclk_freq_hz. A cast to uint32_t is
```

```
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) ((timer_freq_hz * TAU_EXAMPLE_DESIRED_PERIOD_MSEC) /
TAU_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_TAU_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL or period is invalid range. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_UNSUPPORTED | Unsupported for Input pulse Function, High-Low Measurement Function |

### ◆ R_TAU_CompareMatchSet()

fsp_err_t R_TAU_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | TAU compare match is not supported. |

### ◆ R_TAU_DutyCycleSet()

fsp_err_t R_TAU_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

timer_api_t::dutyCycleSet is not supported on the R_TAU Independent channels.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_TAU_InfoGet()

fsp_err_t R_TAU_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
timer_info_t info;
   (void) R_TAU_InfoGet(&g_timer0_ctrl, &info);
   uint32_t timer_freq_hz = info.clock_frequency;
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, frequency, and ELC event written to caller's structure successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_info was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_StatusGet()

fsp_err_t R_TAU_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_TAU_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current timer state and counter value set successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_CallbackSet()

fsp_err_t R_TAU_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_TAU_Close()

fsp_err_t R_TAU_Close ( timer_ctrl_t *const *p_ctrl*)

Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_LinMeasurementFuncSwitch()

fsp_err_t R_TAU_LinMeasurementFuncSwitch ( timer_ctrl_t *const *p_ctrl*, tau_function_t *func* )

This function is provided for the SAU LIN driver to support switching between low level measurement and input pulse measurement functions during LIN reception. It is not intended to be called directly by application code.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to TAU control structure |
| [in] | func | Measurement functions (TAU_FUNCTION_LOW_LEVEL_WIDTH_MEASUREMENT, TAU_FUNCTION_INPUT_PULSE_INTERVAL_MEASUREMENT) |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Switching was successful and measurement has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid input function. |

## 5.2.19.2 Port Output Enable for GPT (r_poeg)

Modules » Timers

**Functions**

| | |
|---|---|
| fsp_err_t | R_POEG_Open (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg) |
| fsp_err_t | R_POEG_OutputDisable (poeg_ctrl_t *const p_ctrl) |
| fsp_err_t | R_POEG_Reset (poeg_ctrl_t *const p_ctrl) |
| fsp_err_t | R_POEG_StatusGet (poeg_ctrl_t *const p_ctrl, poeg_status_t *const p_status) |
| fsp_err_t | R_POEG_CallbackSet (poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void *const p_context, poeg_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_POEG_Close (poeg_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface.

# Overview

The POEG module can be used to configure events to disable GPT GTIOC output pins.

### Features

The POEG module has the following features:

- Supports disabling GPT output pins based on GTETRG input pin level.
- Supports disabling GPT output pins based on comparator crossing events (configurable in the Comparator, High-Speed (r_acmphs) driver).
- Supports disabling GPT output pins when GTIOC pins are the same level (configurable in the Timer, General PWM (r_gpt) driver).
- Supports disabling GPT output pins when main oscillator stop is detected.
- Supports disabling GPT output pins by software API.
- Supports notifying the application when GPT output pins are disabled by POEG.
- Supports resetting POEG status.

# Configuration

### Build Time Configurations for r_poeg

The following build time configurations are defined in fsp_cfg/r_poeg_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Timers > Port Output Enable for GPT (r_poeg)

This module can be added to the Stacks tab via New Stack > Timers > Port Output Enable for GPT (r_poeg). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_poeg0 | Module name. |
| Channel | Must be a valid POEG channel | 0 | Specify the hardware channel. |
| Trigger | MCU Specific Options | | Select the trigger |

| | | | |
|---|---|---|---|
| | | | sources that will enable POEG. Software disable is always supported. This configuration can only be set once after reset. It cannot be modified after the initial setting. |
| Input | | | |
| GTETRG Polarity | • Active High<br>• Active Low | Active High | Select the polarity of the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger. |
| GTETRG Noise Filter | • Disabled<br>• PCLKB/1<br>• PCLKB/8<br>• PCLKB/32<br>• PCLKB/128 | Disabled | Configure the noise filter for the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) when GPT output pins are disabled by POEG. |
| Interrupt Priority | MCU Specific Options | | Select the POEG interrupt priority. |

### Clock Configuration

The POEG clock is based on the PCLKB frequency.

### Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, or GTETRGD as an input signal to disable GPT output pins.

# Usage Notes

### POEG GTETRG Pin and Channel

The POEG channel number corresponds to the GTETRG input pin that can be used with the channel. GTETRGA must be used with POEG channel 0, GTETRGB must be used with POEG channel 1, etc.

### Limitations

The user should be aware of the following limitations when using POEG:

- The POEG trigger source can only be set once per channel. Modifying the POEG trigger source after it is set is not allowed by the hardware.
- The POEG cannot be disabled using this API. The interrupt is disabled in R_POEG_Close(), but the POEG will still disable the GPT output pins if a trigger is detected even if the module is closed.

# Examples

### POEG Basic Example

This is a basic example of minimal use of the POEG in an application.

```
void poeg_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the POEG. */
    err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### POEG Callback Example

This is an example of a using the POEG callback to restore GPT output operation.

```
/* Example callback called when POEG disables GPT output pins. */
void poeg_callback (poeg_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 /* (Optional) Determine the cause of the POEG event. */
 poeg_status_t status;
    (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
 /* Correct the cause of the POEG event before resetting POEG. */
 /* Reset the POEG before exiting the callback. */
    (void) R_POEG_Reset(&g_poeg0_ctrl);
 /* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
  * again. */
```

```
do
   {
     (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
   } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
 /* Alternatively, if the POEG cannot be reset, disable the POEG interrupt to prevent
it from firing continuously.
  * Update the 0 in the macro below to match the POEG channel number. */
   NVIC_DisableIRQ(VECTOR_NUMBER_POEG0_EVENT);
}
```

## Data Structures

| | |
|---|---|
| struct | poeg_instance_ctrl_t |

## Data Structure Documentation

### ◆ poeg_instance_ctrl_t

| struct poeg_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when poeg_api_t::open is called. |

## Function Documentation

### ◆ R_POEG_Open()

fsp_err_t R_POEG_Open ( poeg_ctrl_t *const *p_ctrl*, poeg_cfg_t const *const *p_cfg* )

Initializes the POEG module and applies configurations. Implements poeg_api_t::open.

***Note***

*The poeg_cfg_t::trigger setting can only be configured once after reset. Reopening with a different trigger configuration is not possible.*

Example:

```
/* Initializes the POEG. */
   err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the source divider is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | poeg_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_POEG_OutputDisable()

fsp_err_t R_POEG_OutputDisable ( poeg_ctrl_t *const *p_ctrl*)

Disables GPT output pins. Implements poeg_api_t::outputDisable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | GPT output pins successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_Reset()

fsp_err_t R_POEG_Reset ( poeg_ctrl_t *const *p_ctrl*)

Resets status flags. Implements poeg_api_t::reset.

***Note***

*Status flags are only reset if the original POEG trigger is resolved. Check the status using R_POEG_StatusGet after calling this function to verify the status is cleared.*

Example:

```
/* Correct the cause of the POEG event before resetting POEG. */
/* Reset the POEG before exiting the callback. */
   (void) R_POEG_Reset(&g_poeg0_ctrl);
/* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
 * again. */
do
   {
     (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
   } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Function attempted to clear status flags. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_StatusGet()

fsp_err_t R_POEG_StatusGet ( poeg_ctrl_t *const *p_ctrl*, poeg_status_t *const *p_status* )

Get current POEG status and store it in provided pointer p_status. Implements poeg_api_t::statusGet.

Example:

```
/* (Optional) Determine the cause of the POEG event. */
poeg_status_t status;
   (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current POEG state stored successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_POEG_CallbackSet()

fsp_err_t R_POEG_CallbackSet ( poeg_ctrl_t *const *p_ctrl*, void(*)(poeg_callback_args_t *) *p_callback*, void *const *p_context*, poeg_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements poeg_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_POEG_Close()

| fsp_err_t R_POEG_Close ( poeg_ctrl_t *const *p_ctrl*) |
|---|
| Disables POEG interrupt. Implements poeg_api_t::close.<br><br>***Note***<br>*This function does not disable the POEG.*<br>**Return values** |

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

## 5.2.19.3 Realtime Clock (r_rtc)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_RTC_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg) |
| fsp_err_t | R_RTC_Close (rtc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RTC_ClockSourceSet (rtc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| fsp_err_t | R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| fsp_err_t | R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate) |
| fsp_err_t | R_RTC_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |
| fsp_err_t | R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg) |
| fsp_err_t | R_RTC_CallbackSet (rtc_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_RTC_TimeCaptureSet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture) |
| fsp_err_t | R_RTC_TimeCaptureGet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture) |

### Detailed Description

Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface.

# Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to the alarm and periodic interrupt.

### Features

- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC alarm and periodic event notification.
- RTC time capture.

The RTC HAL module supports three different interrupt types:

- An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
- A periodic interrupt generated every 2, 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, or 1/256 second(s)
- A carry interrupt is used internally when reading time from the RTC calender to get accurant time readings.
  *Note*
  *See "Reading 64-Hz Counter and Time" in the RTC section of the relevant hardware manual for more details.*
  A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt. When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

### Date and Time validation

"Parameter Checking" needs to be enabled if date and time validation is required for calendarTimeSet and calendarAlarmSet APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using the calendarAlarmSet API, only the fields which have their corresponding match flag set are written to the registers. Other register fields are reset to default value.

### Sub-Clock error adjustment (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused

by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low. The time error adjustment functions include:

- Automatic adjustment
- Adjustment by software

The error adjustment is reset every time RTC is reconfigured or time is set.

*Note*

*RTC driver configurations do not do error adjustment internally while initiliazing the driver. Application must make calls to the error adjustment APIs for desired adjustment. See "Time Error Adjustment Function" in the RTC section of the relevant hardware manual for more details on this feature.*

# Configuration

### Build Time Configurations for r_rtc

The following build time configurations are defined in fsp_cfg/r_rtc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Set Source Clock in Open | • Enabled<br>• Disabled | Enabled | If enabled RTC source clock is initialized in R_RTC_Open. If disabled, R_RTC_ClockSourceSet must be called to set the RTC source clock. Disable if user wants to control the setting of RTC source clock after warm start. |

### Configurations for Timers > Realtime Clock (r_rtc)

This module can be added to the Stacks tab via New Stack > Timers > Realtime Clock (r_rtc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rtc0 | Module name. |
| Clock Source | MCU Specific Options | | Select the RTC clock source. |
| Frequency Comparison Value (LOCO) | Value must be a positive integer between 7 and 511 | 255 | Frequency comparison value when using LOCO |

| | | | |
|---|---|---|---|
| Automatic Adjustment Mode | • Enabled<br>• Disabled | Enabled | Enable/ Disable the Error Adjustment mode |
| Automatic Adjustment Period | • 10 Seconds<br>• 1 Minute<br>• NONE | 10 Seconds | Select the Error Adjustment Period for Automatic Adjustment |
| Adjustment Type (Plus-Minus) | • NONE<br>• Addition<br>• Subtraction | NONE | Select the Error Adjustment type |
| Error Adjustment Value | Manual Entry | 0 | Specify the Adjustment Value (the number of sub-clock cycles) from the prescaler |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Alarm Interrupt Priority | MCU Specific Options | | Select the alarm interrupt priority. |
| Period Interrupt Priority | MCU Specific Options | | Select the period interrupt priority. |
| Carry Interrupt Priority | MCU Specific Options | | Select the carry interrupt priority. |

*Note*

*See "Frequency Register (RFRH/RFRL)" in the RTC section of the relevant hardware manual for more details*

### Interrupt Configuration

To activate interrupts for the RTC module, the desired interrupts must be enabled, The underlying implementation will be expected to handle any interrupts it can support and notify higher layers via callback.

### Clock Configuration

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator) with less accuracy
- Sub-clock oscillator with increased accuracy

Users have to select the right source for their application. LOCO is the default during configuration when it is available.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### System Initialization

- RTC driver does not start the sub-clock. The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

- Carry interrupt priority must be set to avoid incorrect time returned from calendarTimeGet API during roll-over.
- Even when only running in Periodic Interrupt mode R_RTC_CalendarTimeSet must be called successfully to start the RTC.
- In case of warm start or a hard reset (when VBATT powers RTC in case of power loss), user might not want to reinitialize the clock source after reset. In this case, disable the Set Source Clock in Open property for RTC and call R_RTC_ClockSourceSet() only when RTC clock source needs to be set. Application should check for the reset type and should accordingly call the R_RTC_ClockSourceSet().

### Limitations

Developers should be aware of the following limitations when using the RTC:

- R_RTC operates in 24-hour mode.
- Binary-count mode is not supported.
- The R_RTC_CalendarTimeGet() cannot be used from an interrupt that has higher priority than the carry interrupt. Also, it must not be called with interrupts disabled globally, as this API internally uses carry interrupt for its processing. API may return incorrect time if this is done.
- Time capture input pins should be configured prior to opening RTC.
- When multiple events are detected, the capture time for the first event is retained. Time capture value must be got and reset status bit for the next capture when having event input.

### VRTC-Domain

VRTC-domain provides power supply to SOSC and RTC on devices with IRTC:

- In case VRTC-domain is invalid, SOSC and RTC will be in undetermined state. Any operation related to RTC and sub-clock should be avoided.
- When VRTC-domain becomes valid again, application is responsible to reinitialize SOSC and RTC. Sub-clock initialization can be done by calling R_BSP_SubclockInitialize().
- BSP provides R_BSP_SubclockStatusGet() to check the status of VRTC domain. But, it is recommended to use LVD to monitor VRTC domain change. Please refer to Low/Programmable Voltage Detection (r_lvd) module for more details.

### Time Capture

The RTC is capable of storing the month, day, hour, minute and second when detecting an edge of a signal on a time capture event input pin. On RA parts up to three capture channels can be configured.

- Use R_RTC_TimeCaptureSet to configure the detection source and noise filter for each capture channel.
- Use R_RTC_TimeCaptureGet to check the latest capture. If no capture has been triggered the function will return FSP_ERR_INVALID_STATE.
- Even when the supply of power from the VCC power supply pin is stopped and VRTC power is being supplied, the time capture function of the time capture event input pins (RTCICn) can be used.

# Examples

### RTC Basic Example

This is a basic example of minimal use of the RTC in an application.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_time =
{
    .tm_sec = 10,
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
    .tm_wday = 3,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
rtc_time_t get_time;
void rtc_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module */
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 /* Set the calendar time */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Get the calendar time */
```

```
 R_RTC_CalendarTimeGet(&g_rtc0_ctrl, &get_time);
}
```

### RTC Clock Source Set Example

This is an example of how to handle call to set the RTC clock.

```
void rtc_clock_source_set_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* This condition can differ based on use case. */
 if (R_SYSTEM->RSTSR0 == 1)
    {
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
    }
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Periodic interrupt example

This is an example of periodic interrupt in RTC.

```
void rtc_periodic_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Alarm interrupt example

This is an example of alarm interrupt in RTC.

```
void rtc_alarm_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
```

```
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 R_RTC_CalendarAlarmSet(&g_rtc0_ctrl, &set_time1);
 /* Wait for the Alarm interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Error Adjustment example

This is an example of modifying error adjustment in RTC.

```
void rtc_erroradj_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
 R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 /* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
 R_RTC_ClockSourceSet(&g_rtc0_ctrl);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 /* Modify Error Adjustment after RTC is running */
    err = R_RTC_ErrorAdjustmentSet(&g_rtc0_ctrl, &err_cfg2);
    assert(FSP_SUCCESS == err);
}
```

### RTC with VRTC-domain example

This is an example of calling RTC API on devices with VRTC-domain.

```
#if BSP_FEATURE_RTC_IS_IRTC
void rtc_vrtc_domain_checking_example ()
{
 fsp_err_t       err       = FSP_SUCCESS;
 static uint32_t rtc_opened = 0;
```

```
 while (1)
    {
       err = R_BSP_SubclockStatusGet();
 if (FSP_SUCCESS != err)
     {
 /* RTC API should not be called. */
 /* Try to initialize SOSC. */
           err = R_BSP_SubclockInitialize();
 if (FSP_SUCCESS == err)
     {
 /* Delay for sub-clock oscillation stabilization time */
 R_BSP_SoftwareDelay(BSP_CLOCK_CFG_SUBCLOCK_STABILIZATION_MS,
BSP_DELAY_UNITS_MILLISECONDS);
     }
     }
 else
     {
 /* VRTC-domain reset will not initialize RTC registers. */
 if (0 == rtc_opened)
     {
               err       = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
               rtc_opened = (FSP_SUCCESS == err) ? 1 : 0;
     }
 /* Other RTC API can be called if needed. */
     }
    }
}
#endif
```

### RTC Time Capture example

This is an example of calling Time Capture API on RTC with Independent Power Supply.

```
void rtc_irtc_time_capture_example ()
{
```

```
fsp_err_t err = FSP_SUCCESS;
/* Open the RTC module*/
   err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Configure a GPIO pin to trigger the Time Capture pin */
R_BSP_PinCfg(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, (uint32_t) BSP_IO_DIRECTION_OUTPUT
);
/* Preset trigger pin */
R_BSP_PinWrite(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, BSP_IO_LEVEL_LOW);
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
/* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
R_RTC_ClockSourceSet(&g_rtc0_ctrl);
/* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
/* Set time capture configuration for the provided channel when RTC is running */
   err = R_RTC_TimeCaptureSet(&g_rtc0_ctrl, &g_rtc_time_capture);
   assert(FSP_SUCCESS == err);
/* Update trigger pin */
R_BSP_PinWrite(RTC_TIME_CAPTURE_TRIGGER_CHANNEL_0, BSP_IO_LEVEL_HIGH);
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
/* Get the time capture for the provided channel */
   err = R_RTC_TimeCaptureGet(&g_rtc0_ctrl, &g_rtc_time_capture);
   assert(FSP_SUCCESS == err);
}
```

### Data Structures

| | |
|---|---|
| struct | rtc_extended_cfg_t |
| struct | rtc_instance_ctrl_t |

## Data Structure Documentation

### ◆ rtc_extended_cfg_t

| struct rtc_extended_cfg_t |
|---|

| RTC extend configuration | | |
|---|---|---|
| Data Fields | | |
| uint8_t | alarm1_ipl | Alarm 1 interrupt priority. |
| IRQn_Type | alarm1_irq | Alarm 1 interrupt vector. |

### ◆ rtc_instance_ctrl_t

| struct rtc_instance_ctrl_t | |
|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called | |
| **Data Fields** | |
| uint32_t | open |
| | Whether or not driver is open. |
| | |
| const rtc_cfg_t * | p_cfg |
| | Pointer to initial configurations. |
| | |
| volatile bool | carry_isr_triggered |
| | Was the carry isr triggered. |
| | |

## Function Documentation

### ◆ R_RTC_Open()

fsp_err_t R_RTC_Open ( rtc_ctrl_t *const *p_ctrl*, rtc_cfg_t const *const *p_cfg* )

Opens and configures the RTC driver module. Implements rtc_api_t::open. Configuration includes clock source, and interrupt callback function.

Example:

```
/* Open the RTC module */
   err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and RTC has started. |
| FSP_ERR_ASSERTION | Invalid p_ctrl or p_cfg pointer. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |

### ◆ R_RTC_Close()

fsp_err_t R_RTC_Close ( rtc_ctrl_t *const *p_ctrl*)

Close the RTC driver. Implements rtc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | De-Initialization was successful and RTC driver closed. |
| FSP_ERR_ASSERTION | Invalid p_ctrl. |
| FSP_ERR_NOT_OPEN | Driver not open already for close. |

### ◆ R_RTC_ClockSourceSet()

fsp_err_t R_RTC_ClockSourceSet ( rtc_ctrl_t *const *p_ctrl*)

Sets the RTC clock source. Implements rtc_api_t::clockSourceSet.

Example:

```
/* This condition can differ based on use case. */
if (R_SYSTEM->RSTSR0 == 1)
   {
/* Set the RTC clock source. Can be skipped if "Set Source Clock in Open" property
is enabled. */
R_RTC_ClockSourceSet(&g_rtc0_ctrl);
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and RTC has started. |
| FSP_ERR_ASSERTION | Invalid p_ctrl or p_cfg pointer. |
| FSP_ERR_NOT_OPEN | Driver is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Invalid clock source. |

### ◆ R_RTC_CalendarTimeSet()

fsp_err_t R_RTC_CalendarTimeSet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time set operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |

### ◆ R_RTC_CalendarTimeGet()

fsp_err_t R_RTC_CalendarTimeGet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Get the calendar time.

**Warning**

Do not call this function from a critical section or from an interrupt with higher priority than the carry interrupt, or the time returned may be inaccurate.

Implements rtc_api_t::calendarTimeGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_CalendarAlarmSet()

fsp_err_t R_RTC_CalendarAlarmSet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Set the calendar alarm time.

Implements rtc_api_t::calendarAlarmSet.

**Precondition**

The calendar counter must be running before the alarm can be set.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time set operation was successful. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_CalendarAlarmGet()

fsp_err_t R_RTC_CalendarAlarmGet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Get the calendar alarm time.

Implements rtc_api_t::calendarAlarmGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_PeriodicIrqRateSet()

fsp_err_t R_RTC_PeriodicIrqRateSet ( rtc_ctrl_t *const *p_ctrl*, rtc_periodic_irq_select_t const *rate* )

Set the periodic interrupt rate and enable periodic interrupt.

Implements rtc_api_t::periodicIrqRateSet

***Note***

*To start the RTC R_RTC_CalendarTimeSet must be called at least once.*

Example:

```
/* Set the periodic interrupt rate to 1 second */
R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The periodic interrupt rate was successfully set. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_InfoGet()

fsp_err_t R_RTC_InfoGet ( rtc_ctrl_t *const *p_ctrl*, rtc_info_t *const *p_rtc_info* )

Set RTC clock source and running status information ad store it in provided pointer p_rtc_info

Implements rtc_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Get information Successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_ErrorAdjustmentSet()

fsp_err_t R_RTC_ErrorAdjustmentSet ( rtc_ctrl_t *const *p_ctrl*, rtc_error_adjustment_cfg_t const *const *err_adj_cfg* )

This function sets time error adjustment

Implements rtc_api_t::errorAdjustmentSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Time error adjustment successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open for operation. |
| FSP_ERR_UNSUPPORTED | The clock source is not sub-clock. |
| FSP_ERR_INVALID_ARGUMENT | Invalid error adjustment value. |

### ◆ R_RTC_CallbackSet()

fsp_err_t R_RTC_CallbackSet ( rtc_ctrl_t *const *p_ctrl*, void(*)(rtc_callback_args_t *) *p_callback*, void *const *p_context*, rtc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements rtc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to RTC control block is NULL or the RTC is not configured to use the internal clock. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_RTC_TimeCaptureSet()

fsp_err_t R_RTC_TimeCaptureSet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture* )

Set time capture configuration for the provided channel.

Implements rtc_api_t::timeCaptureSet

***Note***

*Updating capture settings requires significant software delay. Timing considerations should be carefully considered when calling this function.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Setting for Time capture was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_INVALID_CHANNEL | Invalid input channel set. |
| FSP_ERR_UNSUPPORTED | Hardware not support this feature. |

### ◆ R_RTC_TimeCaptureGet()

fsp_err_t R_RTC_TimeCaptureGet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture* )

Get time capture value of the provided channel.

Implements rtc_api_t::timeCaptureGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Get time capture successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_INVALID_CHANNEL | Invalid input channel get. |
| FSP_ERR_INVALID_STATE | Invalid operation state. |
| FSP_ERR_UNSUPPORTED | Hardware not support this feature. |

## 5.2.19.4 Realtime Clock (r_rtc_c)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_RTC_C_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg) |
| fsp_err_t | R_RTC_C_Close (rtc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RTC_C_ClockSourceSet (rtc_ctrl_t *const p_ctrl) |
| fsp_err_t | R_RTC_C_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| fsp_err_t | R_RTC_C_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| fsp_err_t | R_RTC_C_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_C_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| fsp_err_t | R_RTC_C_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate) |
| fsp_err_t | R_RTC_C_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |

| | |
|---|---|
| fsp_err_t | R_RTC_C_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg) |
| fsp_err_t | R_RTC_C_CallbackSet (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_RTC_C_TimeCaptureSet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture) |
| fsp_err_t | R_RTC_C_TimeCaptureGet (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture) |

### Detailed Description

Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface.

# Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to the alarm and periodic interrupt.

### Features

- RTC_C time and date get and set.
- RTC_C time and date alarm get and set.
- RTC_C alarm and periodic event notification.

The RTC HAL module supports two different interrupt types:

- An alarm interrupt generated on a match of any combination of minute, hour and day of the week.
- A periodic interrupt generated every 1 month, 1 day, 1 hour, 1 minute, 1 second and 1/2 second.

A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt. When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

### Date and Time validation

"Parameter Checking" needs to be enabled if date and time validation is required for calendarTimeSet and calendarAlarmSet APIs. When using the calendarAlarmSet API, minute, hour and day of the week are written to the related registers.

### Sub-Clock error adjustment (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the

clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low.

The error adjustment is reset every time RTC is reconfigured or time is set.

*Note*

*RTC driver configurations do not do error adjustment internally while initializing the driver. Application must make calls to the error adjustment api's for desired adjustment.*

# Configuration

### Build Time Configurations for r_rtc_c

The following build time configurations are defined in fsp_cfg/r_rtc_c_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Timers > Realtime Clock (r_rtc_c)

This module can be added to the Stacks tab via New Stack > Timers > Realtime Clock (r_rtc_c).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_rtc0 | Module name. |
| Operation clock | • FSXP<br>• SOSC/256 (128 Hz) | FSXP | Select the RTC operation clock. If using 'SOSC/256' as the source for RTCCLK, the FSXP source cannot be LOCO in Clock tab. |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR). |
| Interrupt Priority | MCU Specific Options | | Select the period or alarm interrupt priority. |

### Interrupt Configuration

To activate interrupts for the RTC module, the desired interrupts must be enabled, The underlying implementation will be expected to handle any interrupts it can support and notify higher layers via callback.

### Clock Configuration

The RTC HAL module can use the following clock sources(setting in clock page):

- Sub-Clock: SOSC (32.768 kHz)
- SOSC/256 (128 Hz)
- LOCO (32.768 kHz)

Users have to select the right source for their application. Sub-Clock is the default during configuration when it is available.

### Pin Configuration

This module does not use I/O pins.

## Usage Notes

### System Initialization

- RTC driver does not start the sub-clock. The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning
The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

- Even when only running in Periodic Interrupt mode R_RTC_C_CalendarTimeSet must be called successfully to start the RTC.

### Limitations

Developers should be aware of the following limitations when using the RTC_C:

- R_RTC_C operates in 24-hour mode.
- If using 'SOSC/256' as the source for RTC operation clock, the FSXP source cannot be LOCO in Clock tab.
- Time error correction function can only proceed when the setting of the RTCC0.RTC128EN bit is 0.
- Alarm clock interrupt can work under LOCO clock source, but it is unstable. Please use alarm clock interrupt function under other clocks.

## Examples

### RTC Basic Example

This is a basic example of minimal use of the RTC_C in an application.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_time =
{
    .tm_sec = 10,
```

```
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
    .tm_wday = 3,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
rtc_time_t get_time;
void rtc_c_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module */
    err = R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the calendar time */
 R_RTC_C_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Get the calendar time */
 R_RTC_C_CalendarTimeGet(&g_rtc0_ctrl, &get_time);
}
```

### RTC Calendar Time Set example

This is an example of setting calendar time in RTC_C.

Warning

The weekday must be set correctly even when parameter checking is disabled.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_calendar_time =
{
    .tm_sec = 10,
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
 /* tm_wday value must be set correctly */
```

```
    .tm_wday = 5,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
void rtc_c_calendar_time_set_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module */
    err = R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the calendar time and date */
    err = R_RTC_C_CalendarTimeSet(&g_rtc0_ctrl, &set_calendar_time);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

### RTC Periodic interrupt example

This is an example of periodic interrupt in RTC.

```
void rtc_c_periodic_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the RTC module*/
    err = R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* R_RTC_C_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_C_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_C_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
```

```
 /* Wait for interrupt */
    }
}
```

### RTC Alarm interrupt example

This is an example of alarm interrupt in RTC.

```
void rtc_c_alarm_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
    err = R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 R_RTC_C_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 R_RTC_C_CalendarAlarmSet(&g_rtc0_ctrl, &set_time1);
 /* Wait for the Alarm interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

### RTC Error Adjustment example

This is an example of modifying error adjustment in RTC.

```
void rtc_c_erroradj_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Open the RTC module*/
 R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 R_RTC_C_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Modify Error Adjustment after RTC is running */
    err = R_RTC_C_ErrorAdjustmentSet(&g_rtc0_ctrl, &err_cfg2);
```

```
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | rtc_c_extended_cfg |
| struct | rtc_c_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rtc_c_subclock_division_t |

## Data Structure Documentation

### ◆ rtc_c_extended_cfg

| struct rtc_c_extended_cfg | | |
|---|---|---|
| RTC extended configuration | | |
| Data Fields | | |
| rtc_c_subclock_division_t | clock_source_div | The sub clock division value. |

### ◆ rtc_c_instance_ctrl_t

| struct rtc_c_instance_ctrl_t | |
|---|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called | |
| **Data Fields** | |
| uint32_t | open |
| | Whether or not driver is open. |
| | |
| const rtc_cfg_t * | p_cfg |
| | Pointer to initial configurations. |
| | |

## Enumeration Type Documentation

### ◆ rtc_c_subclock_division_t

| enum rtc_c_subclock_division_t | |
|---|---|
| RTC sub_clock division | |
| Enumerator | |
| RTC_CLOCK_SOURCE_SUBCLOCK_DIV_BY_1 | Using sub_clock for rct_c clock. |
| RTC_CLOCK_SOURCE_SUBCLOCK_DIV_BY_256 | Using (sub_clock / 256) for rct_c clock. |

## Function Documentation

### ◆ R_RTC_C_Open()

fsp_err_t R_RTC_C_Open ( rtc_ctrl_t *const *p_ctrl*, rtc_cfg_t const *const *p_cfg* )

Opens and configures the RTC driver module. Implements rtc_api_t::open. Configuration includes clock source, and interrupt callback function.

R_RTC_Open should be called to manipulate the RTC instead of setting register directly.

Example:

```
/* Open the RTC module */
   err = R_RTC_C_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and RTC has started. |
| FSP_ERR_UNSUPPORTED | Invalid clock source. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_INVALID_MODE | Subsystem clock should use SOSC before setting realtime clock to SOSC/256. |
| FSP_ERR_ASSERTION | Invalid time parameter field. |

### ◆ R_RTC_C_Close()

fsp_err_t R_RTC_C_Close ( rtc_ctrl_t *const *p_ctrl*)

Close the RTC driver. Implements rtc_api_t::close

**Return values**

| | |
|---|---|
| FSP_SUCCESS | De-Initialization was successful and RTC driver closed. |
| FSP_ERR_ASSERTION | Invalid p_ctrl. |
| FSP_ERR_NOT_OPEN | Driver not open already for close. |

### ◆ R_RTC_C_ClockSourceSet()

fsp_err_t R_RTC_C_ClockSourceSet ( rtc_ctrl_t *const *p_ctrl*)

Setting clock source is not supported on the RTC_C.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Please set clock source in clock page. |

### ◆ R_RTC_C_CalendarTimeSet()

fsp_err_t R_RTC_C_CalendarTimeSet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time set operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_INVALID_ARGUMENT | Invalid parameter field. |

### ◆ R_RTC_C_CalendarTimeGet()

fsp_err_t R_RTC_C_CalendarTimeGet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time* )

Get the calendar time.

**Warning**

Do not call this function from a critical section or from an interrupt with higher priority than the carry interrupt, or the time returned may be inaccurate.

Implements rtc_api_t::calendarTimeGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_C_CalendarAlarmSet()

fsp_err_t R_RTC_C_CalendarAlarmSet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Set the calendar alarm time. For the p_alarm, only minutes, hours and weekdays are valid. Minutes 0 to 59. Hours 0 to 23. Weekdays 0 to 127.

Implements rtc_api_t::calendarAlarmSet.

**Precondition**

The calendar counter must be running before the alarm can be set.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time set operation was successful. |
| FSP_ERR_INVALID_ARGUMENT | Invalid time parameter field. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | Interrupt must be enabled to use the alarm. |

### ◆ R_RTC_C_CalendarAlarmGet()

fsp_err_t R_RTC_C_CalendarAlarmGet ( rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm* )

Get the calendar alarm time. For the p_alarm, only minutes, hours and weekdays will be got.

Implements rtc_api_t::calendarAlarmGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Calendar alarm time get operation was successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |

### ◆ R_RTC_C_PeriodicIrqRateSet()

fsp_err_t R_RTC_C_PeriodicIrqRateSet ( rtc_ctrl_t *const *p_ctrl*, rtc_periodic_irq_select_t const *rate* )

Set the periodic interrupt rate and enable periodic interrupt.

Implements rtc_api_t::periodicIrqRateSet

***Note***

*To start the RTC R_RTC_C_CalendarTimeSet must be called at least once.*

Example:

```
/* Set the periodic interrupt rate to 1 second */
R_RTC_C_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The periodic interrupt rate was successfully set. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open already for operation. |
| FSP_ERR_IRQ_BSP_DISABLED | User IRQ parameter not valid |

### ◆ R_RTC_C_InfoGet()

fsp_err_t R_RTC_C_InfoGet ( rtc_ctrl_t *const *p_ctrl*, rtc_info_t *const *p_rtc_info* )

Get RTC_C running status information ad store it in provided pointer p_rtc_info

Implements rtc_api_t::infoGet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Get information Successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Realtime clock module is stopped. |

### ◆ R_RTC_C_ErrorAdjustmentSet()

fsp_err_t R_RTC_C_ErrorAdjustmentSet ( rtc_ctrl_t *const *p_ctrl*, rtc_error_adjustment_cfg_t const *const *err_adj_cfg* )

This function sets time error adjustment

Implements rtc_api_t::errorAdjustmentSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Time error adjustment successful. |
| FSP_ERR_ASSERTION | Invalid input argument. |
| FSP_ERR_NOT_OPEN | Driver not open for operation. |
| FSP_ERR_INVALID_ARGUMENT | Not under sub-clock or invalid error adjustment value. |

### ◆ R_RTC_C_CallbackSet()

fsp_err_t R_RTC_C_CallbackSet ( rtc_ctrl_t *const *p_ctrl*, void(*)(rtc_callback_args_t *) *p_callback*, void *const *p_context*, rtc_callback_args_t *const *p_callback_memory* )

Updates the user callback and has option of providing memory for callback structure. Implements rtc_api_t::callbackSet

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Baud rate was successfully changed. |
| FSP_ERR_ASSERTION | Pointer to RTC control block is NULL or the RTC is not configured to use the internal clock. |
| FSP_ERR_NOT_OPEN | The control block has not been opened |

### ◆ R_RTC_C_TimeCaptureSet()

fsp_err_t R_RTC_C_TimeCaptureSet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture* )

Capture is not supported on the RTC_C.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_RTC_C_TimeCaptureGet()

fsp_err_t R_RTC_C_TimeCaptureGet ( rtc_ctrl_t *const *p_ctrl*, rtc_time_capture_t *const *p_time_capture* )

Capture is not supported on the RTC_C.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

## 5.2.19.5 Three-Phase PWM (r_gpt_three_phase)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg) |
| fsp_err_t | R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle) |
| fsp_err_t | R_GPT_THREE_PHASE_CallbackSet (three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface.

# Overview

The General PWM Timer (GPT) Three-Phase driver provides basic functionality for synchronously starting and stopping three PWM channels for use in 3-phase motor control applications. A function is additionally provided to allow setting duty cycle values for all three channels, optionally with double-buffering.

### Features

The GPT Three-Phase driver provides the following functions:

- Synchronize configuration of three GPT channels
- Synchronously start, stop and reset all three GPT channels
- Set duty cycle on all three channels with one function

# Configuration

### Build Time Configurations for r_gpt_three_phase

The following build time configurations are defined in fsp_cfg/r_gpt_three_phase_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Timers > Three-Phase PWM (r_gpt_three_phase)

This module can be added to the Stacks tab via New Stack > Timers > Three-Phase PWM (r_gpt_three_phase). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_three_phase0 | Module name. |
| Mode | • Triangle-Wave Symmetric PWM<br>• Triangle-Wave Asymmetric PWM<br>• Triangle-Wave Asymmetric PWM (Mode 3) | Triangle-Wave Symmetric PWM | Mode selection. Triangle-Wave Symmetric PWM: Generates symmetric PWM waveforms with duty cycle determined by compare match set during a crest interrupt and updated at the next trough. Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest. |
| Period | Value must be a non-negative integer less than or equal to 0x40000000000 | 15 | Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period.<br><br>If the requested period cannot be achieved, the settings with the largest possible period |

| | | | |
|---|---|---|---|
| | | | that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structures for each timer. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Kilohertz | Unit of the period specified above |
| GPT U-Channel | Value must be an integer greater than or equal to 0 | 0 | Specify the GPT channel for U signal output. |
| GPT V-Channel | Value must be an integer greater than or equal to 0 | 1 | Specify the GPT channel for V signal output. |
| GPT W-Channel | Value must be an integer greater than or equal to 0 | 2 | Specify the GPT channel for W signal output. |
| Callback Channel | • U-Channel<br>• V-Channel<br>• W-Channel | U-Channel | Specify the GPT channel to set a callback for when using R_GPT_THREE_PHASE_CallbackSet. |
| Buffer Mode | • Single Buffer<br>• Double Buffer | Single Buffer | When Double Buffer is selected the 'duty_buffer' array in three_phase_duty_cycle_t is used as a buffer for the 'duty' array. This allows setting the duty cycle for the next two crest/trough events in asymmetric mode with only one call to R_GPT_THREE_PHASE_DutyCycleSet. |
| GTIOCA Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| GTIOCB Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |

Extra Features

Extra Features > Dead Time

| | | | |
|---|---|---|---|
| Dead Time Count Up (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff). | 0 | Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants. |
| Dead Time Count Down (Raw Counts) (GPTE/GPTEH only) | Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff). | 0 | Select the dead time to apply during down counting. This value only applies to the GPT32E and GPT32EH variants. |

### Clock Configuration

Please refer to the Timer, General PWM (r_gpt) section for more information.

### Pin Configuration

Please refer to the Timer, General PWM (r_gpt) section for more information.

# Usage Notes

Warning

Be sure the GTIOCA/B stop level and dead time values are set appropriately for your application before attempting to drive a motor. Failure to do so may result in damage to the motor drive circuitry and/or the motor itself if the timer is stopped by software.

### Initial Setup

The following should be configured once the GPT Three-Phase module has been added to a project:

1. Set "Pin Output Support" in one of the GPT submodules to "Enabled with Extra Features"
2. Configure common settings in the GPT Three-Phase module properties
3. Set the crest and trough interrupt priority and callback function in **one** of the three GPT submodules (if desired)
4. Set the "Extra Features -> Output Disable" settings in each GPT submodule as needed for your application

*Note*

*Because all three modules are operated synchronously with the same period interrupts only need to be enabled in one of the three GPT modules.*

### Buffer Modes

There are two buffering modes available for duty cycle values - single- and double-buffered. In single buffer mode only the values specified in the duty array element of three_phase_duty_cycle_t are used by R_GPT_THREE_PHASE_DutyCycleSet. At the next trough or crest event the output duty cycle will be internally updated to the set values.

In double buffer mode the duty_buffer array values are used as buffer values for the duty elements. Once passed to R_GPT_THREE_PHASE_DutyCycleSet, the next trough or crest event will update the output duty cycle to the values specified in duty as before. However, at the following crest or trough event the output duty cycle will be updated to the values in duty_buffer. This allows the duty cycle for both sides of an asymmetric PWM waveform to be set at only one trough or crest event per period instead of at every event.

# Examples

### GPT Three-Phase Basic Example

This is a basic example of minimal use of the GPT Three-Phase module in an application. The duty cycle is updated at every timer trough with the previously loaded buffer value, then the duty cycle buffer is reloaded in the trough interrupt callback.

```
void gpt_callback (timer_callback_args_t * p_args)
{
 fsp_err_t                err;
 three_phase_duty_cycle_t duty_cycle;
 if (TIMER_EVENT_TROUGH == p_args->event)
    {
 /* Update duty cycle values (example) */
       duty_cycle.duty[THREE_PHASE_CHANNEL_U] =
get_duty_counts(THREE_PHASE_CHANNEL_U);
       duty_cycle.duty[THREE_PHASE_CHANNEL_V] =
get_duty_counts(THREE_PHASE_CHANNEL_V);
       duty_cycle.duty[THREE_PHASE_CHANNEL_W] =
get_duty_counts(THREE_PHASE_CHANNEL_W);
 /* Update duty cycle values */
       err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Handle crest event. */
    }
}
void gpt_three_phase_basic_example (void)
{
```

```
fsp_err_t err = FSP_SUCCESS;
/* Initializes the module. */
   err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Start the timer. */
   (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | gpt_three_phase_instance_ctrl_t |

## Data Structure Documentation

### ◆ gpt_three_phase_instance_ctrl_t

| struct gpt_three_phase_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when three_phase_api_t::open is called. |

## Function Documentation

### ◆ R_GPT_THREE_PHASE_Open()

fsp_err_t R_GPT_THREE_PHASE_Open ( three_phase_ctrl_t *const *p_ctrl*, three_phase_cfg_t const *const *p_cfg* )

Initializes the 3-phase timer module (and associated timers) and applies configurations. Implements three_phase_api_t::open.

Example:

```
/* Initializes the module. */
   err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful. |
| FSP_ERR_ASSERTION | A required input pointer is NULL. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |

### ◆ R_GPT_THREE_PHASE_Stop()

fsp_err_t R_GPT_THREE_PHASE_Stop ( three_phase_ctrl_t *const *p_ctrl*)

Stops all timers synchronously. Implements three_phase_api_t::stop.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timers successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_Start()

fsp_err_t R_GPT_THREE_PHASE_Start ( three_phase_ctrl_t *const *p_ctrl*)

Starts all timers synchronously. Implements three_phase_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timers successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_Reset()

fsp_err_t R_GPT_THREE_PHASE_Reset ( three_phase_ctrl_t *const *p_ctrl*)

Resets the counter values to 0. Implements three_phase_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counters were reset successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_THREE_PHASE_DutyCycleSet()

fsp_err_t R_GPT_THREE_PHASE_DutyCycleSet ( three_phase_ctrl_t *const *p_ctrl*, three_phase_duty_cycle_t *const *p_duty_cycle* )

Sets duty cycle for all three timers. Implements three_phase_api_t::dutyCycleSet.

In symmetric PWM mode duty cycle values are reflected after the next trough. In asymmetric PWM mode values are reflected at the next trough OR crest, whichever comes first.

When double-buffering is enabled the values in three_phase_duty_cycle_t::duty_buffer are set to the double-buffer registers. When values are reflected the first time the single buffer values (three_phase_duty_cycle_t::duty) are used. On the second reflection the duty_buffer values are used. In asymmetric PWM mode this enables both count-up and count-down PWM values to be set at trough (or crest) exclusively.

***Note***

*It is recommended to call this function in a high-priority callback to ensure that it is not interrupted and that no GPT events occur during setting that would result in a duty cycle buffer load operation.*

Example:

```
/* Update duty cycle values */
      err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
      assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | One or more duty cycle count values was outside the range 0..(period - 1). |

### ◆ R_GPT_THREE_PHASE_CallbackSet()

fsp_err_t R_GPT_THREE_PHASE_CallbackSet ( three_phase_ctrl_t *const *p_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback for the GPT U-channel with the option to provide memory for the callback argument structure. Implements three_phase_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_GPT_THREE_PHASE_Close()

fsp_err_t R_GPT_THREE_PHASE_Close ( three_phase_ctrl_t *const *p_ctrl*)

Stops counters, disables output pins, and clears internal driver data. Implements three_phase_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

## 5.2.19.6 Timer, 32-bit Interval Timer (r_tml)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_TML_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_TML_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TML_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TML_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TML_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TML_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TML_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_TML_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_TML_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_TML_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_TML_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |

| | |
|---|---|
| fsp_err_t | R_TML_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_TML_Close (timer_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the TML peripherals on RA MCUs. This module implements the Timer Interface.

# Overview

The TML module can be used to count or count input events. TML operates with the HOCO, MOCO, MOSC, LOCO/SOSC clock or the event input from the ELC, which is asynchronous to CPU operation.

### Features

The TML module has the following features:

- Supports 8-bit counter mode: 4 independent channels (Channel 0,Channel 1,Channel 2 and Channel 3).
- Supports 16-bit counter mode: 2 independent channels (Channel 0 and Channel 2).
- Supports 32-bit counter mode: 1 channel (channel 0).
- Supports 16-bit capture mode: 1 channel (channel 0).
- Supports count source of HOCO, MOCO, MOSC, LOCO/SOSC clock or ELC events.
- Configurable period (counts per timer cycle).
- Supports runtime reconfiguration of period.
- Supports capture-start by external sources from ELC events.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status.

### Selecting a Timer

RA MCUs have different timer peripherals: the General PWM Timer (GPT), the 32-bit Interval Timer (TML), Timer Array Unit (TAU), and the Asynchronous General Purpose Timer (AGT). Some MCUs have multiple timer modules. When selecting between them, consider these factors:

| | GPT | TML | AGT | TAU |
|---|---|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The TML can operate in all low power modes. | The AGT can operate in all low power modes. | The TAU can not operate in all low power modes. |
| Available Channels | The number of GPT channels is device specific. Currently supported MCUs have at least 7 GPT channels. | Currently supported MCUs have 4 TML channels. | Currently supported MCUs have 2 AGT channels. | Currently supported MCUs have 8 TAU channels. |
| Timer Resolution | Currently | The TML timers | The AGT timers | The TAU timers |

| | | | | |
|---|---|---|---|---|
| | supported MCUs have at least one 32-bit GPT timer. | are 32-bit/16-bit/8-bit timers. | are 16-bit timers. | are 16-bit/8-bit timers. |
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The TML runs off MOSC, LOCO/SOSC, HOCO, MOCO with a configurable divider up to 128 or ELC events. | The AGT runs off PCLKB, LOCO, or subclock. | The TAU runs off PCLK with a configurable divider up to 32768 |

# Configuration

### Build Time Configurations for r_tml

The following build time configurations are defined in fsp_cfg/r_tml_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Critical Section Guarding | • Enabled<br>• Disabled | Disabled | Enable critical section guarding around peripheral configuration updates. This should be enabled if multiple instances of the R_TML module are being used. |
| 16-bit Capture Mode Support | • Disabled<br>• Enabled | Disabled | Controls whether 16-bit capture mode support is included in the build. This setting applies globally to all r_tml_instances. If 16-bit capture mode is not used by any instance, disable this setting to reduce ROM usage. |
| Enable Single Channel | • Disabled<br>• Enabled | Disabled | Enable single channel to reduce code size. Assumes channel 0. |
| Interrupt Support | • Disabled<br>• Enabled | Disabled | Enable support for interrupts. |

### Configurations for Timers > 32-bit Interval Timer (r_tml)

This module can be added to the Stacks tab via New Stack > Timers > 32-bit Interval Timer (r_tml).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Mode | • 8-bit Counter Mode<br>• 16-bit Counter Mode<br>• 32-bit Counter Mode<br>• 16-bit Capture Mode | 16-bit Counter Mode | Selection of Operating mode. |
| Channel Selection | Channel number must exist on this MCU | 0 | Specify the hardware channel. |
| ELC event | MCU Specific Options | | Select the elc event. |
| Counter Mode Settings | | | |
| Period | Value must be positive integer | 0x10000 | Specify the timer period in units selected below.<br><br>Set the period to 0x100 (8-bit) or 0x10000 (16-bit) or 0x100000000 (32-bit) raw counts for a free running timer. Since the maxmimum divider is 128... the raw counts period can be set up to the (max count+1)*128. [0x8000 (8-bit) or 0x800000 (16-bit), or 0x8000000000 (32-bit)]. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above. If the MCU supports ELC and the Counter Clock Source is set to ELC, only Raw Counts can be selected here. |
| Capture Mode Settings | | | |
| Capture Mode Settings > 16-Bit Counter Input Settings (when Capture source = Interrupt on compare match with ITLCMP01) | | | |
| Period | Value must be positive | 0x10000 | Specify the timer |

| | | | |
|---|---|---|---|
| | integer | | period in units selected below.<br><br>Set the period to 0x10000 raw counts in the maximum period of 16-bit capture. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above. If MCU support ELC and the Capture Clock Source is the ELC, only Raw Counts can be selected here. |
| Capture Trigger | MCU Specific Options | | Selection of capture trigger. |
| Capture Clock Divider | • fITL0<br>• fITL0/2<br>• fITL0/4<br>• fITL0/8<br>• fITL0/16<br>• fITL0/32<br>• fITL0/64<br>• fITL0/128 | fITL0 | Configure FDIV0 for the counter clock in 16-bit Capture Mode. |
| Counter Status | • Retained after the completion of capturing<br>• Cleared after the completion of capturing | Retained after the completion of capturing | Selection of the 16-bit counter (ITL000 + ITL001) clearing after capturing. |
| Interrupt | | | |
| Callback function | Name must be a valid C symbol | NULL | A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers |
| Priority | MCU Specific Options | | Select the interrupt priority. |

### Clock Configuration

The TML operating clock is based on the HOCO, MOCO, MOSC, or LOCO/SOSC. The operating frequency will be based on the source chosen and the divider selected. You can set the clock frequencies using the **Clocks** tab of the RA Configuration editor or change the clock divider for the capture mode channel by **Capture Mode Settings->Capture Clock Divider**.

*Note*

*The TML channels may be alternatively configured to use an event input from the ELC as a count clock in case no frequency/duration can be input, only "Raw Counts".*

### Pin Configuration

# Usage Notes

### Maximum Period

In capture mode, the clock divider for the capture channel has to be selected in **Capture Mode Settings->Capture Clock Divider**. If ITLCMP01 is the capture trigger input, the clock divider for channels 2 + 3 is not available.

When no capture mode is active, the RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x8000000000/0x800000/0x8000 on a 32-bit/16-bit/8-bit timer. This will configure the timer with the maximum period and a count clock divisor of 128.

*Note*

*When an event input from ELC is used as an operation clock, the automatic calculation of the period count value and source clock divider is not available.*

### Updating Period

The period is updated after calling R_TML_PeriodSet().

*Note*

*The period is only updated when the counter is stopped.*
*When using the API to set the period counts, the maximum timer_cfg_t::period_counts for 8-bit and 16-bit modes are 0x100 and 0x10000, respectively. To set the the maximum count for 32-bit mode, set the timer_cfg_t::period_counts to 0 (as 0x10000000 is outside the 32-bit range).*
*In the 16-bit capture mode, if the interrupt on compare match with ITLCMP01 is used as trigger source, the period counts of the 16-bit timer channels 2 + 3 can be changed at run-time.*

### Capture mode

When the 16-bit capture mode is to be used for channels 0 and 1, the counter value is stored in interval timer capture register 00 (ITLCAP00) in response to the selected capture trigger.

*Note*

*When Channels 0 and 1 are in capture mode, Channels 2 and 3 can be used to trigger the capture when ITLCMP01 is the capture trigger. In this case the counter setup for Channels 2 and 3 are in **Capture Mode Settings->16-Bit Counter Input Settings**. If ITLCMP01 is not the capture trigger input, Channels 2 and 3 may be setup as an independent instance of 16-bit Counter channel.*
*When a capture trigger is generated when the timer is not running, the previous counter value of the capture channel is still copied to the Interval Timer Capture Register (ITLCAP00).*
*The **Common->16-bit Capture Mode Support** property in the module configuration must be enabled to use the capture operation.*

### Controlling TML with ELC Events

The TML timer can be configured to start count down or trigger capture when an ELC event occurs.

*Note*

*The configurable ELC TML sources are shared by all TML channels.*
*The event links for the ELC must be configured outside this module.*

### Triggering ELC Events with TML

The TML timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

# Examples

### TML Basic Example

This is a basic example of minimal use of the TML in an application.

```
void tml_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TML_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TML_Start(&g_timer0_ctrl);
}
```

### TML Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### TML Counter Example

This is an example of 8/16/32-bit Counter.

```
void tml_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TML_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TML_Start(&g_timer0_ctrl);
 /* Read the current state. Counter value is in status.state. */
 timer_status_t status;
    (void) R_TML_StatusGet(&g_timer0_ctrl, &status);
 /* (Optional) Enable the mask for the interrupt flag. */
    (void) R_TML_Disable(&g_timer0_ctrl);
 /* (Optional) Disable the mask to use the interrupt. */
    (void) R_TML_Enable(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_TML_Stop(&g_timer0_ctrl);
}
```

### TML Capture Example

This is an example of using the TML to capture the counter value of configured channel.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CAPTURE_EDGE == p_args->event)
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
```

```
      (void) R_TML_InfoGet(&g_timer0_ctrl, &info);
      uint32_t period = info.period_counts;
      g_captured_time    = (period * g_capture_overflows) + p_args->capture;
      g_capture_overflows = 0U;
   }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
   {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
      g_capture_overflows++;
   }
}
void tml_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
   err = R_TML_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
 /* There are 4 ways to trigger the capture:
  * - Use software trigger by set the ITLCC0.CAPR bit to 1 directly after call start
  * - Use ELC event and wait for valid input
  * - Use LOCO/SOSC clock and wait for valid edge
  * - Use interrupt on compare match with ITLCPM01
  *
  * This example uses the last option.
  */
   (void) R_TML_Start(&g_timer0_ctrl);
}
```

### TML Period Update Example

This an example of updating the period.

```
#define TML_EXAMPLE_MSEC_PER_SEC (1000)
```

```
#define TML_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void tml_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TML_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Get the source clock frequency (in Hz) */
 timer_info_t info;
    (void) R_TML_InfoGet(&g_timer0_ctrl, &info);
    uint32_t timer_freq_hz = info.clock_frequency;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX. A cast to uint64_t is used to prevent
this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * TML_EXAMPLE_DESIRED_PERIOD_MSEC) /
TML_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_TML_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TML_Start(&g_timer0_ctrl);
}
```

## Data Structures

| | |
|---|---|
| struct | tml_extended_cfg_t |
| struct | tml_instance_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | tml_clock_t |
| enum | tml_capture_trigger_t |

| | |
|---|---|
| enum | tml_counter_status_t |

## Data Structure Documentation

### ◆ tml_extended_cfg_t

| struct tml_extended_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| tml_capture_trigger_t | capture_trigger | Select the capture source for capture channel. |
| tml_counter_status_t | counter_status | Status of 16-bit counter (ITL000 + ITL001) after completion of capturing. |

### ◆ tml_instance_ctrl_t

| struct tml_instance_ctrl_t | |
|---|---|
| TML instance control block. | |
| **Data Fields** | |
| uint32_t | open |
| | Used to determine if the channel is configured. |
| | |
| const timer_cfg_t * | p_cfg |
| | Pointer to the configuration block. |
| | |
| uint8_t | channel_mask |
| | Mask value of channel used. |
| | |
| void(* | p_callback )(timer_callback_args_t *) |
| | Pointer to callback that is called when a timer event occurs. |
| | |
| void * | p_context |
| | Pointer to context to be passed into callback function. |
| | |

## Enumeration Type Documentation

### ◆ tml_clock_t

| enum tml_clock_t | |
|---|---|
| Enumeration for TML FITL0, FITL1 clock source | |
| Enumerator | |
| TML_CLOCK_HOCO | HOCO. |
| TML_CLOCK_MOCO | MOCO. |
| TML_CLOCK_MOSC | MOSC. |
| TML_CLOCK_LOCO_SOSC | SOSC. |
| TML_CLOCK_ELC_EVENT | Event input from the ELC. |

### ◆ tml_capture_trigger_t

| enum tml_capture_trigger_t | |
|---|---|
| Enumeration for TML FITL2 capture trigger source | |
| Enumerator | |
| TML_CAPTURE_TRIGGER_SOFTWARE | Software trigger. |
| TML_CAPTURE_TRIGGER_ITLCMP01 | Interrupt on compare match with ITLCMP01. |
| TML_CAPTURE_TRIGGER_LOCO_SOSC | LOCO/SOSC (rising edge) |
| TML_CAPTURE_TRIGGER_EVENT_ELC | Event input from ELC (rising edge) |

### ◆ tml_counter_status_t

| enum tml_counter_status_t | |
|---|---|
| Enumeration for status of 16-bit counter (ITL000 + ITL001) after completion of capturing | |
| Enumerator | |
| TML_COUNTER_STATUS_RETAINED_AFTER_CAPTURING | 16-bit counter (ITL000 + ITL001) is retained after the completion of capturing |
| TML_COUNTER_STATUS_CLEARED_AFTER_CAPTURING | 16-bit counter (ITL000 + ITL001) is cleared after the completion of capturing |

## Function Documentation

### ◆ R_TML_Open()

fsp_err_t R_TML_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the timer module and applies configurations. Implements timer_api_t::open.

TML hardware does not support one-shot functionality natively. If one shot mode is desired, the user code should stop the timer after the timer expires the first time in an ISR after the requested period has elapsed.

The TML implementation of the general timer can accept a tml_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_TML_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the source divider is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |
| FSP_ERR_INVALID_CHANNEL | Selected channel is invalid |
| FSP_ERR_INVALID_MODE | The mode requested in the p_cfg parameter is incorrect. It must be the same for all instances. |
| FSP_ERR_IN_USE | Channel is running |

### ◆ R_TML_Stop()

fsp_err_t R_TML_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops the counter and disable the capture. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_TML_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_Start()

fsp_err_t R_TML_Start ( timer_ctrl_t *const *p_ctrl*)

Starts the counter and enable the capture. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_TML_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_Reset()

fsp_err_t R_TML_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_Enable()

fsp_err_t R_TML_Enable ( timer_ctrl_t *const *p_ctrl*)

Enable the interrupt generation from the selected channel timer_api_t::enable.

Example:

```
/* (Optional) Disable the mask to use the interrupt. */
   (void) R_TML_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_Disable()

fsp_err_t R_TML_Disable ( timer_ctrl_t *const *p_ctrl*)

Disable the interrupt generation for this timer. Implements timer_api_t::disable.

***Note***

*The timer could be stop after R_TML_Disable().*

Example:

```
/* (Optional) Enable the mask for the interrupt flag. */
   (void) R_TML_Disable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_PeriodSet()

fsp_err_t R_TML_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Sets period value provided. Only set this value when all timers are stop. Implements timer_api_t::periodSet.

Example:

```
/* Get the source clock frequency (in Hz) */
timer_info_t info;
   (void) R_TML_InfoGet(&g_timer0_ctrl, &info);
   uint32_t timer_freq_hz = info.clock_frequency;
/* Calculate the desired period based on the current clock. Note that this calculation could overflow if the
 * desired period is larger than UINT32_MAX. A cast to uint64_t is used to prevent this. */
   uint32_t period_counts =
      (uint32_t) (((uint64_t) timer_freq_hz * TML_EXAMPLE_DESIRED_PERIOD_MSEC) / TML_EXAMPLE_MSEC_PER_SEC);
/* Set the calculated period. */
   err = R_TML_PeriodSet(&g_timer0_ctrl, period_counts);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IN_USE | Channel is running |

### ◆ R_TML_DutyCycleSet()

fsp_err_t R_TML_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

timer_api_t::dutyCycleSet is not supported on the TML.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_TML_CompareMatchSet()

fsp_err_t R_TML_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

timer_api_t::compareMatchSet is not supported on the TML.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | Function not supported in this implementation. |

### ◆ R_TML_InfoGet()

fsp_err_t R_TML_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Get timer configuration information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_TML_InfoGet(&g_timer0_ctrl, &info);
      uint32_t period = info.period_counts;
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, frequency, and ELC event written to caller's structure successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_info was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TML_StatusGet()

fsp_err_t R_TML_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current state. Counter value is in status.state. */
timer_status_t status;
   (void) R_TML_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current timer state and counter value set successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_TML_CallbackSet()

fsp_err_t R_TML_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_TML_Close()

| fsp_err_t R_TML_Close ( timer_ctrl_t *const *p_ctrl*) |
|---|
| Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Successful close.<br>FSP_ERR_ASSERTION — p_ctrl was NULL.<br>FSP_ERR_NOT_OPEN — The instance is not opened.<br>FSP_ERR_IP_CHANNEL_NOT_PRESENT — The channel requested in the p_cfg parameter is not available on this device.<br>FSP_ERR_IN_USE — Channel is running |

## 5.2.19.7 Timer, General PWM (r_gpt)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_GPT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_GPT_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_GPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_GPT_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_GPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |

| | |
|---|---|
| fsp_err_t | R_GPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_GPT_CounterSet (timer_ctrl_t *const p_ctrl, uint32_t counter) |
| fsp_err_t | R_GPT_OutputEnable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin) |
| fsp_err_t | R_GPT_OutputDisable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin) |
| fsp_err_t | R_GPT_AdcTriggerSet (timer_ctrl_t *const p_ctrl, gpt_adc_compare_match_t which_compare_match, uint32_t compare_match_value) |
| fsp_err_t | R_GPT_PwmOutputDelaySet (timer_ctrl_t *const p_ctrl, gpt_pwm_output_delay_edge_t edge, gpt_pwm_output_delay_setting_t delay_setting, uint32_t const pin) |
| fsp_err_t | R_GPT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_GPT_Close (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_GPT_PwmOutputDelayInitialize () |

### Detailed Description

Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface.

# Overview

The GPT module can be used to count events, measure external input signals, generate a periodic interrupt, or output a periodic or PWM signal to a GTIOC pin.

This module supports the GPT peripherals GPT32EH, GPT32E, GPT32, and GPT16. GPT16 is a 16-bit timer. The other peripherals (GPT32EH, GPT32E, and GPT32) are 32-bit timers. The 32-bit timers are all treated the same in this module from the API perspective.

### Features

The GPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Supports count source of PCLK, GTETRG pins, GTIOC pins, or ELC events.
- Supports debounce filter on GTIOC pins.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Supports runtime reconfiguration of period.

- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports runtime reconfiguration of compare match value.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports start, stop, clear, count up, count down, and capture by external sources from GTETRG pins, GTIOC pins, or ELC events.
- Supports symmetric and asymmetric PWM waveform generation.
- Supports One shot synchronous pulse waveform generation.
- Supports automatic addition of dead time.
- Supports generating ELC events to start an ADC scan at a compare match value (see Event Link Controller (r_elc)) and updating the compare match value.
- Supports linking with a POEG channel to automatically disable GPT output when an error condition is detected.
- Supports setting the counter value while the timer is stopped.
- Supports enabling and disabling output pins.
- Supports skipping up to seven overflow/underflow (crest/trough) interrupts at a time
- Supports generating custom PWM waveforms by configuring the pin's output level at each compare match and cycle end.
- Supports GTIOCnA and GTIOCnB input/output polarity control.

### Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

| | GPT | AGT |
|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The AGT can operate in all low power modes. |
| Available Channels | The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels. | All MCUs have 2 AGT channels. |
| Timer Resolution | All MCUs have at least one 32-bit GPT timer. | The AGT timers are 16-bit timers. |
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The AGT runs off PCLKB, LOCO, or subclock. |

# Configuration

### Build Time Configurations for r_gpt

The following build time configurations are defined in fsp_cfg/r_gpt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP) | Default (BSP) | If selected code for |

| | | | |
|---|---|---|---|
| | • Enabled<br>• Disabled | | parameter checking is included in the build. |
| Pin Output Support | • Disabled<br>• Enabled<br>• Enabled with Extra Features | Disabled | Enables or disables support for outputting PWM waveforms on GTIOCx pins. The "Enabled with Extra Features" option enables support for Triangle wave modes and also enables the features located in the "Extra Features" section of each module instance. |
| Write Protect Enable | • Enabled<br>• Disabled | Disabled | If selected write protection is applied to all GPT channels. |

**Configurations for Timers > Timer, General PWM (r_gpt)**

This module can be added to the Stacks tab via New Stack > Timers > Timer, General PWM (r_gpt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| General > Compare Match | | | |
| General > Compare Match > Compare Match A | | | |
| Status | • Enabled<br>• Disabled | Disabled | |
| Compare match value | Value must be a non-negative integer less than or equal to 0x40000000000 | 0 | Specify the compare match A value in units that selected in Period Unit section. |
| General > Compare Match > Compare Match B | | | |
| Status | • Enabled<br>• Disabled | Disabled | |
| Compare match value | Value must be a non-negative integer less than or equal to 0x40000000000 | 0 | Specify the compare match B value in units that selected in Period Unit section. |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Channel | Enter the supported Channel number | 0 | Specify the hardware channel. |

| | | | |
|---|---|---|---|
| Mode | • Periodic<br>• One-Shot<br>• One-Shot Pulse<br>• Saw-wave PWM<br>• Triangle-wave PWM (symmetric, Mode 1)<br>• Triangle-wave PWM (asymmetric, Mode 2)<br>• Triangle-wave PWM (asymmetric, Mode 3) | Periodic | Mode selection.<br>Periodic: Generates periodic interrupts or square waves.<br>One-shot: Generate a single interrupt or a pulse wave. Note: One-shot mode is implemented in software. ISRs must be enabled for one-shot even if callback is unused.<br>One-Shot Pulse: Counter performs saw-wave operation with fixed buffer operation.<br>Saw-wave PWM: Generates basic saw-wave PWM waveforms.<br>Triangle-wave PWM (symmetric, Mode 1): Generates symmetric PWM waveforms with duty cycle determined by compare match set with 32-bit transfer during a crest event and updated at the next trough with single or double buffer operation.<br>Triangle-wave PWM (asymmetric, Mode 2): Generates asymmetric PWM waveforms with duty cycle determined by compare match set with 32-bit transfer during a crest/trough event and updated at the next trough/crest.<br>Triangle-wave PWM (asymmetric, Mode 3): Generates PWM waveforms with duty cycle determined by compare match set with 64-bit transfer during a crest interrupt and updated at the next trough with fixed buffer operation. |
| Period | Value must be a non- | 0x10000 | Specify the timer |

| | | | |
|---|---|---|---|
| | negative integer less than or equal to 0x40000000000 | | period in units selected below. Set the period to 0x100000000 (32-bit) or 0x10000 (16-bit) raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000 (32-bit) or 0x4000000 (16-bit) which will use a divider of 1024 with the maximum period.<br><br>If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structure. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Output | | | |
| Output > Custom Waveform | | | |
| Output > Custom Waveform > GTIOA | | | |
| Initial Output Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Set the initial output level of GTIOCxA. |
| Cycle End Output Level | • Pin Level Retain<br>• Pin Level Low<br>• Pin Level High<br>• Pin Level Toggle | Pin Level Retain | Set the output level of GTIOCxA at cycle end. |
| Compare Match Output Level | • Pin Level Retain<br>• Pin Level Low<br>• Pin Level High<br>• Pin Level Toggle | Pin Level Retain | Set the output level of GTIOCxA at compare match. |
| Retain Output Level at | • Disabled | Disabled | Retain the current |

| | | | |
|---|---|---|---|
| Count Stop | • Enabled | | GTIOxA output level when counting is stopped. |
| Output > Custom Waveform > GTIOB | | | |
| Initial Output Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Set the initial output level of GTIOCxB. |
| Cycle End Output Level | • Pin Level Retain<br>• Pin Level Low<br>• Pin Level High<br>• Pin Level Toggle | Pin Level Retain | Set the output level of GTIOCxB at cycle end. |
| Compare Match Output Level | • Pin Level Retain<br>• Pin Level Low<br>• Pin Level High<br>• Pin Level Toggle | Pin Level Retain | Set the output level of GTIOCxB at compare match. |
| Retain Output Level at Count Stop | • Disabled<br>• Enabled | Disabled | Retain the current GTIOxB output level when counting is stopped. |
| Custom Waveform Enable | • Disabled<br>• Enabled | Disabled | Enable custom waveform configuration. |
| Duty Cycle Percent (only applicable in PWM mode) | Value must be between 0 and 100 | 50 | Specify the timer duty cycle percent. Only used in PWM mode. |
| GTIOCA Output Enabled | • True<br>• False | False | Enable the output of GTIOCA on a pin. |
| GTIOCA Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| GTIOCB Output Enabled | • True<br>• False | False | Enable the output of GTIOCB on a pin. |
| GTIOCB Stop Level | • Pin Level Low<br>• Pin Level High | Pin Level Low | Select the behavior of the output pin when the timer is stopped. |
| Input | | | |
| Count Up Source | MCU Specific Options | | Select external source that will increment the counter. If any count up source is selected, the timer will count the external sources only. It will not count PCLKD cycles. |
| Count Down Source | MCU Specific Options | | Select external source |

| | | | |
|---|---|---|---|
| | | | that will decrement the counter. If any count down source is selected, the timer will count the external sources only. It will not count PCLKD cycles. |
| Start Source | MCU Specific Options | | Select external source that will start the timer.<br><br>For pulse width measurement, set the Start Source and the Clear Source to the trigger edge (the edge to start the measurement), and set the Stop Source and Capture Source (either A or B) to the opposite edge (the edge to stop the measurement).<br><br>For pulse period measurement, set the Start Source, the Clear Source, and the Capture Source (either A or B) to the trigger edge (the edge to start the measurement). |
| Stop Source | MCU Specific Options | | Select external source that will stop the timer. |
| Clear Source | MCU Specific Options | | Select external source that will clear the timer. |
| Capture A Source | MCU Specific Options | | Select external source that will trigger a capture A event. |
| Capture B Source | MCU Specific Options | | Select external source that will trigger a capture B event. |
| Noise Filter A Sampling Clock Select | • No Filter<br>• Filter PCLKD / 1<br>• Filter PCLKD / 4<br>• Filter PCLKD / 16<br>• Filter PCLKD / 64 | No Filter | Select the input filter for GTIOCA. |
| Noise Filter B Sampling Clock Select | • No Filter<br>• Filter PCLKD / 1 | No Filter | Select the input filter for GTIOCB. |

| | | | |
|---|---|---|---|
| | • Filter PCLKD / 4<br>• Filter PCLKD / 16<br>• Filter PCLKD / 64 | | |
| Pin Polarity | | | |
| GTIOCnA Polarity | • Normal<br>• Inverted | Normal | Input/output polarity control for GTIOCnA pin. |
| GTIOCnB Polarity | • Normal<br>• Inverted | Normal | Input/output polarity control for GTIOCnB pin. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses |
| Overflow/Crest Interrupt Priority | MCU Specific Options | | Select the overflow interrupt priority. This is the crest interrupt for triangle-wave PWM. |
| Capture/Compare match A Interrupt Priority | MCU Specific Options | | Select the interrupt priority for Capture/Compare match A. |
| Capture/Compare match B Interrupt Priority | MCU Specific Options | | Select the interrupt priority for Capture/Compare match B. |
| Underflow/Trough Interrupt Priority | MCU Specific Options | | Select the interrupt priority for the trough interrupt (triangle-wave PWM only). |
| Extra Features | | | |
| Extra Features > Output Disable | | | |
| POEG Link | MCU Specific Options | | Select which POEG to link this GPT channel to. |
| Output Disable POEG Trigger | • Dead Time Error<br>• GTIOCA and | | Select which errors send an output disable trigger to POEG. Dead |

| | | | |
|---|---|---|---|
| | GTIOCB High Level<br>• GTIOCA and GTIOCB Low Level | | time error is only available GPT channels that have GTINTAD.GRPDTE. |
| GTIOCA Disable Setting | • Disable Prohibited<br>• Set Hi Z<br>• Level Low<br>• Level High | Disable Prohibited | Select the disable setting for GTIOCA. |
| GTIOCB Disable Setting | • Disable Prohibited<br>• Set Hi Z<br>• Level Low<br>• Level High | Disable Prohibited | Select the disable setting for GTIOCB. |
| Extra Features > ADC Trigger | | | |
| Start Event Trigger (Channels with GTINTAD only) | • Trigger Event A/D Converter Start Request A During Up Counting<br>• Trigger Event A/D Converter Start Request A During Down Counting<br>• Trigger Event A/D Converter Start Request B During Up Counting<br>• Trigger Event A/D Converter Start Request B During Down Counting | | Select which A/D converter start request interrupts to generate and at which point in the cycle to generate them. |
| ADC A Compare Match (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff). | 0 | Select the compare match value that generates a GPTn AD TRIG A event. |
| ADC B Compare Match (Raw Counts) | Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff). | 0 | Select the compare match value that generates a GPTn AD TRIG B event. |
| Extra Features > Dead Time (Value range varies with Channel) | | | |
| Dead Time Count Up (Raw Counts) | Must be a valid non-negative integer with a maximum configurable | 0 | Select the dead time to apply during up counting. This value |

| | | | |
|---|---|---|---|
| | value of 4294967295 (0xffffffff). | | also applies during down counting for channels that do not have GTDVD. The dead time count up value can be set up to 0xffffffff (32-bit) or 0xffff (16-bit). |
| Dead Time Count Down (Raw Counts) (Channels with GTDVD only) | Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff). | 0 | Select the dead time to apply during down counting. The dead time count down value can be set up to 0xffffffff (32-bit) or 0xffff (16-bit). |
| Extra Features > Interrupt Skipping (Channels with GTITC only) | | | |
| Interrupt to Count | • None<br>• Overflow and Underflow (sawtooth)<br>• Crest (triangle)<br>• Trough (triangle) | None | Select the count source for interrupt skipping. The interrupt skip counter increments after each source event. All crest/overflow and trough/underflow interrupts are skipped when the interrupt skip counter is non-zero. |
| Interrupt Skip Count | • 0<br>• 1<br>• 2<br>• 3<br>• 4<br>• 5<br>• 6<br>• 7 | 0 | Select the number of interrupts to skip. |
| Skip ADC Events | • None<br>• ADC A Compare Match<br>• ADC B Compare Match<br>• ADC A and B Compare Match | None | Select ADC events to suppress when the interrupt skip count is not zero. |
| Extra Features | • Enabled<br>• Disabled | Disabled | Select whether to enable extra features on this channel. |

### Clock Configuration

The GPT clock is based on the PCLKD frequency. You can set the PCLKD frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, GTETRGD, GTIOCA and GTIOCB pins as count sources.

This module can use GTIOCA and GTIOCB pins as output pins for periodic or PWM signals.

This module can use GTIOCA and GTIOCB as input pins to measure input signals.

## Usage Notes

### Maximum Period

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x40000000000 on a 32-bit timer or 0x4000000 on a 16-bit timer. This will configure the timer with the maximum period and a count clock divisor of 1024.

*Note*

*16-bit channels inherently have a reduced maximum period compared to 32-bit channels. When setting period values of hundreds of milliseconds or more on 16-bit channels be sure to check the generated output to confirm the actual configured value as it will clip much earlier than 32-bit channels. In general, it is recommended to use a lower-power timer like AGT or RTC if long hardware delays are required.*

### Updating Period and Duty Cycle

The period and duty cycle are updated after the next counter overflow after calling R_GPT_PeriodSet() or R_GPT_DutyCycleSet(). To force them to update before the next counter overflow, call R_GPT_Reset() while the counter is running.

*Note*

*When manually changing the timer period counts the maximum value for a 32-bit GPT is 0x100000000. This number overflows the 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.*
*When changing the period and duty cycle at the same time when Parameter Checking is enabled, call R_GPT_PeriodSet first followed by R_GPT_DutyCycleSet to ensure input validation is performed correctly.*

### One-Shot Mode

The GPT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Transfer (r_dtc).

### One-Shot Mode Output

The output waveform in one-shot mode is one PCLKD cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

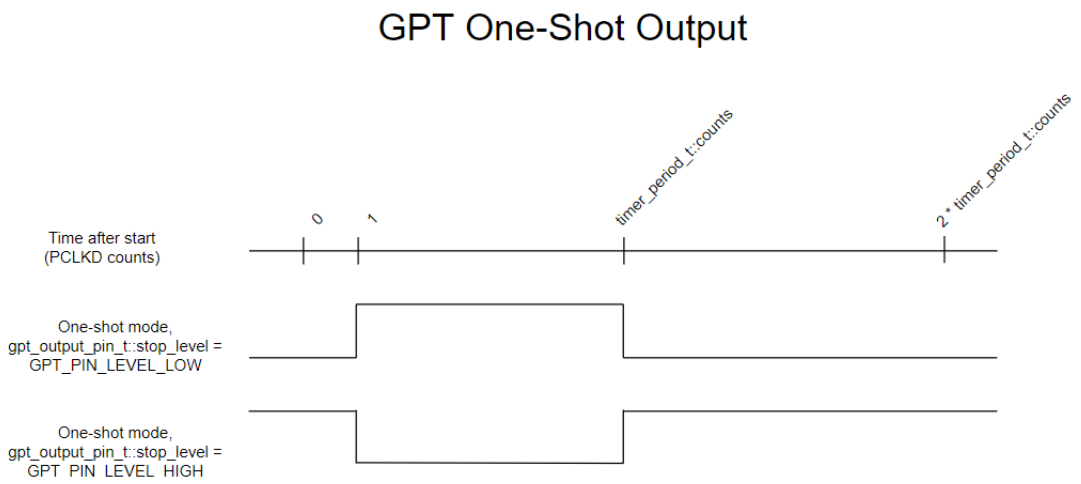



Figure 336: GPT One-Shot Output

### One-Shot Pulse Mode

The one-shot pulse mode is an asymmetric PWM mode that provides more control over the rising and falling edges of the output. The user provides a period and initial output level and controls the signal by specifying compare match values for the leading and trailing edges each period.

*Note*

*Despite its name, the One-Shot Pulse Mode operates continuously and does not stop after the first period.*

### One-Shot Pulse Mode Output

Examples of PWM signals that can be generated by this module are shown below. The leading and trailing edge match values can be modified using R_GPT_DutyCycleSet() in the overflow interrupt.

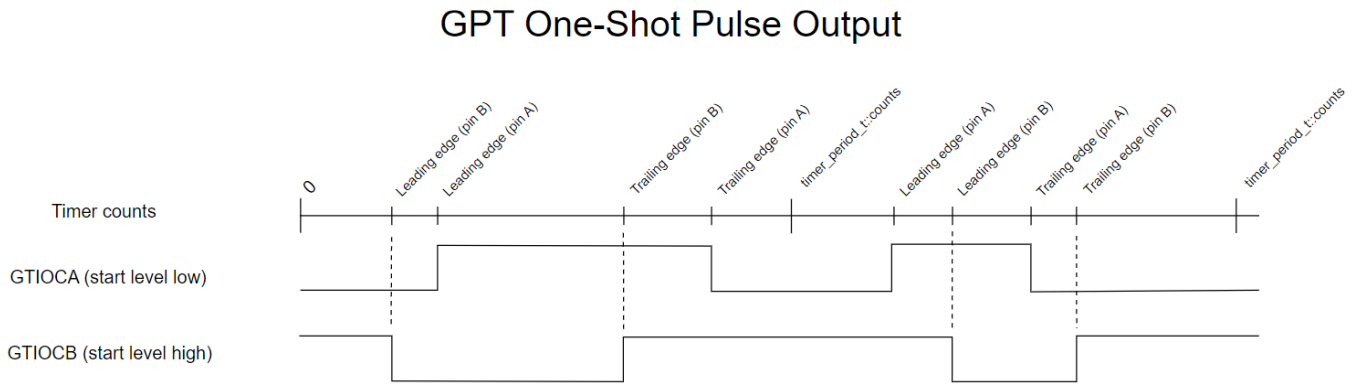



Figure 337: GPT One-Shot Pulse Output (without dead time)

If dead time is enabled only match values for GTIOCnA need to be set; the match values for GTIOCnB will be automatically configured in hardware.

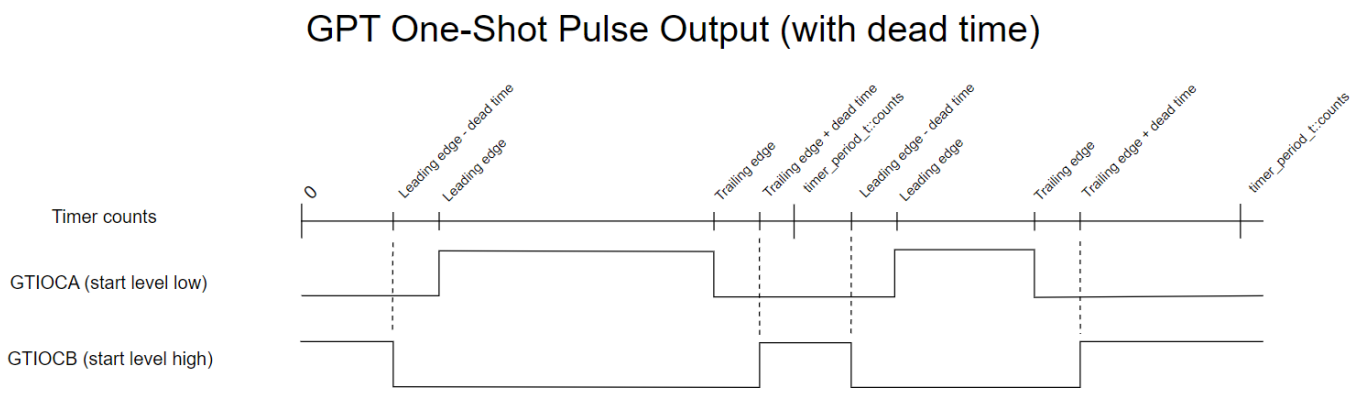



Figure 338: GPT One-Shot Pulse Output (with dead time)

### Periodic Output

The GTIOC pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square wave (from rising edge to rising edge) matches the period of the GPT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

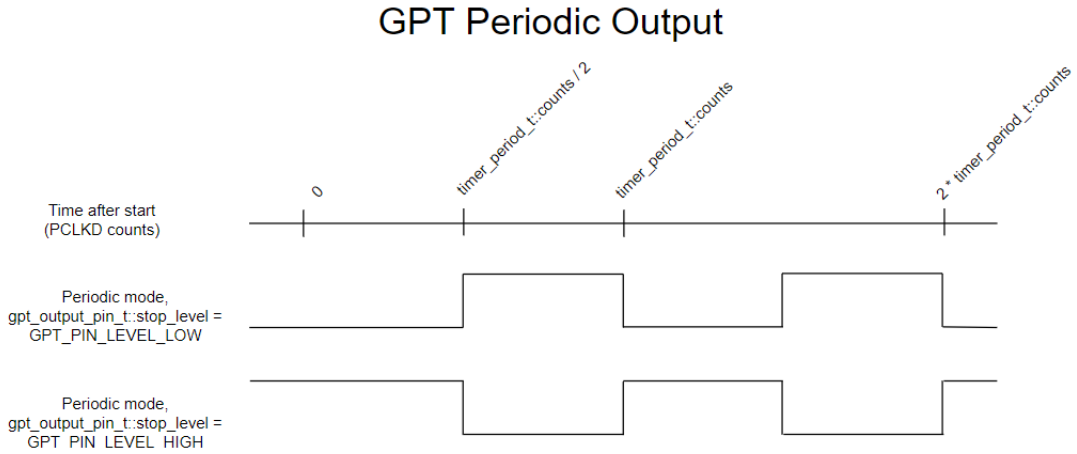



Figure 339: GPT Periodic Output

### PWM Output

The PWM output signal is high at the beginning of the cycle and low at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

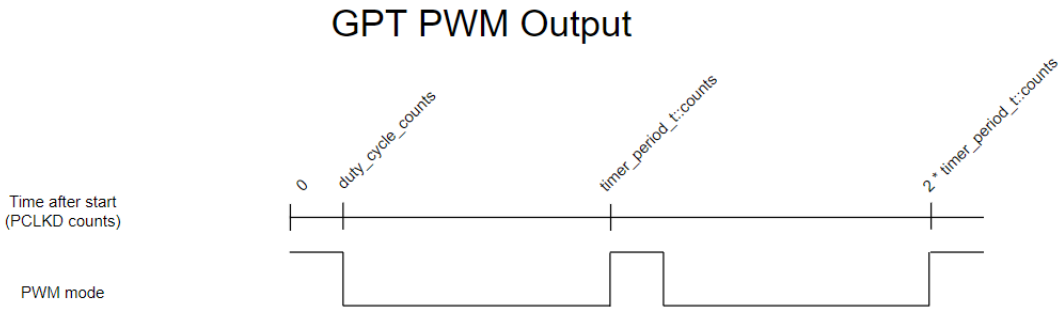



Figure 340: GPT PWM Output

### Triangle-Wave PWM Output

Examples of PWM signals that can be generated by this module are shown below. The duty_cycle_counts can be modified using R_GPT_DutyCycleSet() in the crest interrupt and updated at the following trough for symmetric PWM or modified in both the crest/trough interrupts and updated at the following trough/crest for asymmetric PWM.

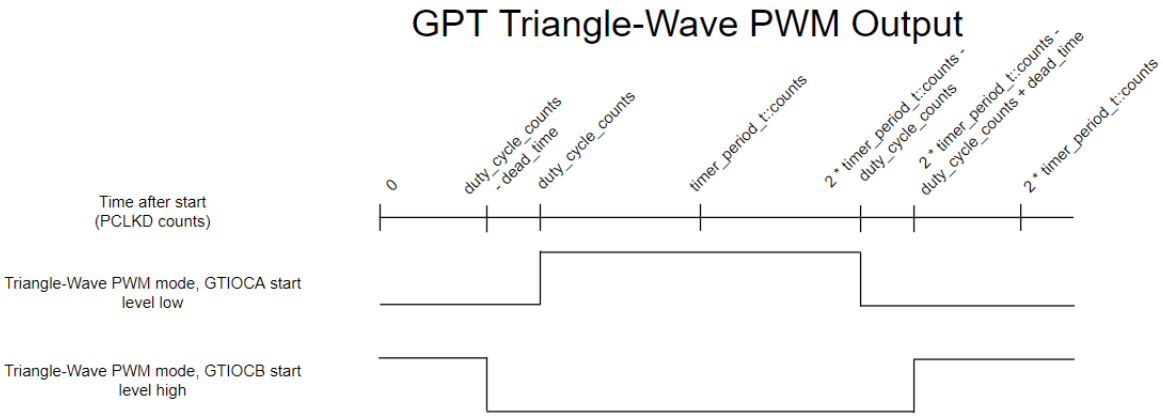



Figure 341: GPT Triangle-Wave PWM Output

### PWM Output Delay Circuit

On select MCUs, an additional PWM output delay circuit can be configured in order to fine tune the rising and falling edge delays in increments of 1/32 times the period of the GPT core clock. The PWM output delay function must be configured prior to initializing the GPT channels using R_GPT_PwmOutputDelayInitialize.

*Note*

*1. In Saw-wave PWM mode, the output delay setting cannot be changed while the capture compare setting (GTCCRn) is greater than or equal the period setting (GTPR) - 2.*
*2. In Triangle PWM modes, the output delay setting cannot be changed while the counter is counting down, and the capture compare setting (GTCCRn) is less than or equal to 2.*
*3. When the PWM Output Delay Circuit is enabled, the PWM signal is delayed by 3 GPT core clock cycles.*
*4. When the GPTCLK is used as the GPT core clock, the following delay is required between writes to the rising or falling edge output delay setting for a given pin: Write_Interval[ns] = Period_of_PCLKA [ns] × 6 + Period_of_GPTCLK [ns] × 4).*

### Event Counting

Event counting can be done by selecting up or down counting sources from GTETRG pins, ELC events, or GTIOC pins. In event counting mode, the GPT counter is not affected by PCLKD.

*Note*

*In event counting mode, the application must call R_GPT_Start() to enable event counting. The counter will not change after calling R_GPT_Start() until an event occurs.*

### Pulse Measurement

If the capture edge occurs before the start edge in pulse measurement, the first capture is invalid (0).

### Controlling GPT with GTETRG Edges

The GPT timer can be configured to stop, start, clear, count up, or count down when a GTETRG rising or falling edge occurs.

*Note*

*The GTETRG pins are shared by all GPT channels.*
*GTETRG pins require POEG to be on (example code for this is provided in GPT Free Running Counter Example).*
*If input filtering is required on the GTETRG pins, that must also be handled outside this module.*

### Controlling GPT with ELC Events

The GPT timer can be configured to stop, start, clear, count up, or count down when an ELC event occurs.

*Note*

*The configurable ELC GPT sources are shared by all GPT channels.*
*The event links for the ELC must be configured outside this module.*

### Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

### Enabling External Sources for Start, Stop, Clear, or Capture

R_GPT_Enable() must be called when external sources are used for start, stop, clear, or capture.

### Interrupt Skipping

When an interrupt skipping source is selected a hardware counter will increment each time the selected event occurs. Each interrupt past the first (up to the specified skip count) will be suppressed. If ADC events are selected for skipping they will also be suppressed except during the timer period leading to the selected interrupt skipping event (see below diagram).

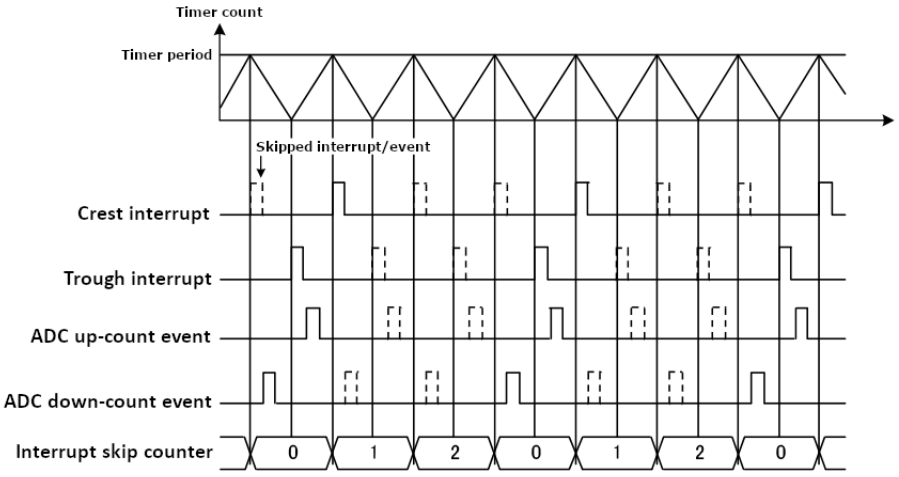



Figure 342: Crest interrupt skipping in triangle-wave PWM modes (skip count 2)

### Complementary Output

By using the Custom Waveform option the output pins can be made to output complementary waveforms. To ensure these waveforms stay in sync, the duty cycle for both pins can be set simultaneously by calling R_GPT_DutyCycleSet once with a pin parameter of GPT_IO_PIN_GTIOCA_AND_GTIOCB.

*Note*

*The pin level for 0% and 100% duty cycle is determined by the Cycle End Output Level in normal PWM mode and the Initial Output Level in triangle PWM modes. 100% duty will output the configured level and 0% will output the opposite. Do not use Pin Level Toggle or Pin Level Retain for the Cycle End Output Level if normal PWM waveforms are desired.*

## Examples

### GPT Basic Example

This is a basic example of minimal use of the GPT in an application.

```
void gpt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
```

### GPT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### GPT Free Running Counter Example

To use the GPT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF for a 32-bit timer or 0xFFFF for a 16-bit timer.

```
void gpt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If event count mode is used to count edges on a GTETRG pin, POEG must
be started to use GTETRG.
  * Reference Note 1 of Table "GPT functions" in the relevant hardware manual. */
 R_BSP_MODULE_START(FSP_IP_POEG, 0U);
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
```

```
 timer_status_t status;
    (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
}
```

### GPT Input Capture Example

This is an example of using the GPT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if ((TIMER_EVENT_CAPTURE_A == p_args->event) || (TIMER_EVENT_CAPTURE_B ==
p_args->event))
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
       uint64_t period = info.period_counts;
 /* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
  * timer_info_t::period_counts. */
 if (0U == period)
      {
            period = UINT32_MAX + 1U;
      }
       g_captured_time    = (period * g_capture_overflows) + p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
```

```
    }
}
void gpt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_GPT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_GPT_Disable(&g_timer0_ctrl);
}
```

### GPT Period Update Example

This an example of updating the period.

```
#define GPT_EXAMPLE_MSEC_PER_SEC (1000)
#define GPT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void gpt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
```

```
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

### GPT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define GPT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void gpt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
}
```

### GPT A/D Converter Start Request Example

This is an example of using the GPT to start the ADC at a configurable A/D converter compare match value.

```
#if ((1U << GPT_EXAMPLE_CHANNEL) & (BSP_FEATURE_GPT_GPTE_CHANNEL_MASK |
BSP_FEATURE_GPT_GPTEH_CHANNEL_MASK | \
  BSP_FEATURE_GPT_AD_DIRECT_START_CHANNEL_MASK))
/* This example shows how to configure the GPT to generate an A/D start request at an
A/D start request compare
 * match value. This example can only be used with GPTE or GPTEH variants. */
void gpt_adc_start_request_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 #if ((BSP_FEATURE_GPT_GPTE_SUPPORTED | BSP_FEATURE_GPT_GPTEH_SUPPORTED) &&
```

```
!BSP_FEATURE_GPT_AD_DIRECT_START_SUPPORTED)
 /* Initialize and configure the ELC. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure the ELC to start a scan on ADC unit 0 when GPT channel 0. Note: This is
typically configured in
  * g_elc_cfg and already set during R_ELC_Open. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0, ELC_EVENT_GPT0_AD_TRIG_A);
    assert(FSP_SUCCESS == err);
 /* Globally enable ELC events. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
 #else  /* BSP_FEATURE_GPT_AD_DIRECT_START_SUPPORTED */
 /* Do not configure AD ELC trigger for devices with GPT ADC direct-start support */
 #endif
 /* Initialize the ADC to start a scan based on an GPT compare-match event trigger.
Set adc_cfg_t::trigger to
  * ADC_TRIGGER_SYNC_ELC. */
    err = p_adc_api->open(&g_adc0_ctrl, &g_adc0_cfg);
    assert(FSP_SUCCESS == err);
    err = p_adc_api->scanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable synchronous triggers by calling R_ADC_ScanStart(). */
    (void) p_adc_api->scanStart(&g_adc0_ctrl);
 /* Initializes the GPT module. Configure gpt_extended_pwm_cfg_t::adc_trigger to set
when the A/D start request
  * is generated. Set gpt_extended_pwm_cfg_t::adc_a_compare_match to set the desired
compare match value. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start the timer. A/D converter start request events are generated each time the
counter is equal to the
  * A/D start request compare match value. */
    (void) R_GPT_Start(&g_timer0_ctrl);
```

```
}
#endif
```

### GPT One-Shot Pulse Mode Example

This example demonstrates the configuration and use of one-shot pulse mode with GPT timer.

```
/* Example callback called when timer overflows. */
void gpt_overflow_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
       g_timer_cycle_end_counter++;
 /* Use R_GPT_DutyCycleSet() API to set new values for each cycle.
  * - Use GPT_IO_PIN_ONE_SHOT_LEADING_EDGE to set the leading edge transition match
value (GTCCRC or GTCCRE register).
  * - Use GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE to set the trailing edge transition match
value (GTCCRD or GTCCRF register).
  */
    }
}
/* GTIOCA (configured for active-high) first and second pulse will be:
 * - 1/2 period, offset by 3/8 period
 * - 1/4 period, offset by 1/4 period */
#define GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_A ((g_timer0_cfg.period_counts * 3) / 8)
#define GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_A ((g_timer0_cfg.period_counts * 7) / 8)
#define GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_A ((g_timer0_cfg.period_counts * 1) / 4)
#define GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_A ((g_timer0_cfg.period_counts * 2) / 4)
/* GTIOCB (configured for active-low) first and second pulse will be:
 * - 1/2 period, offset by 1/4 period
 * - 1/4 period, offset by 1/2 period */
#define GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_B ((g_timer0_cfg.period_counts * 1) / 4)
#define GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_B ((g_timer0_cfg.period_counts * 3) / 4)
#define GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_B ((g_timer0_cfg.period_counts * 2) / 4)
#define GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_B ((g_timer0_cfg.period_counts * 3) / 4)
```

```
/* Configure and output a one-shot pulse that resembles what is shown in the example
above. */
void gpt_one_shot_pulse_mode_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    assert(FSP_SUCCESS == err);
 /* Set first edge transition count for GTIOCA and GTIOCB.
  *
  * Notes:
  * - Two temporary registers (A and B) are used for the second edge.
  * - Temporary register A and Temporary register B may only be loaded by shifting
from GTCCRD and GTCCRF, which is why the GPT_BUFFER_FORCE_PUSH is necessary
  * - If GPT_BUFFER_FORCE_PUSH is not used, timing for the first pulse will be
undefined.
  * - For more information, see "Example setting for saw-wave one-shot pulse mode" in
the UM for your MCU. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FIRST_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Set third edge transition count for GTIOCA and GTIOCB
  * Also force-push to shift all four counts into active buffer locations. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_SECOND_EDGE_PIN_B,
```

```
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE | GPT_BUFFER_FORCE_PUSH);
    assert(FSP_SUCCESS == err);
 /* Set third leading edge transition count for GTIOCA and GTIOCB.
  * Note: It is not necessary to set third and fourth transition edges if it is not
required by the application. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_THIRD_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_LEADING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Set fourth edge transition count for GTIOCA and GTIOCB
  * Do not force-push, shifting third and fourth edges would overwrite active buffer
locations (first and second edge counts). */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_A,
 GPT_IO_PIN_GTIOCA | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl,
                             GPT_ONE_SHOT_EXAMPLE_FOURTH_EDGE_PIN_B,
 GPT_IO_PIN_GTIOCB | GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
 R_GPT_Start(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Wait for the one-shot pulse output to complete then stop GPT
  *
  * Notes:
  * - FSP driver does not currently support automatically stopping one-shot pulse. If
GPT is not stopped the last waveform timing will be repeated indefinitely.
  * - Some (more recent) MCUs added a new register (GTPC) that could be used to
automatically stop GPT after a set number of periods. Software driver support for
```

```
this is scheduled to be added in the future.
  * - The last transition edge timings will continue to be used indefinitely until
the user application stops GPT operation. */
 while (g_timer_cycle_end_counter < 2)
    {
 /* Wait for one-shot pulse output to complete (two pulses) */
    }
 /* (Optional) Stop the timer. */
    err = R_GPT_Stop(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### GPT Compare Match Set Example

This example demonstrates the configuration and use of compare match with GPT timer.

```
/* Example callback called when compare match occurs. */
void gpt_compare_match_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_COMPARE_A == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
#define GPT_COMPARE_MATCH_EXAMPLE_VALUE (0x2000U)
void gpt_compare_match_set_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the compare match value (GPT_COMPARE_MATCH_EXAMPLE_VALUE). This value must be
less than or equal to period value. */
    err = R_GPT_CompareMatchSet(&g_timer0_ctrl, GPT_COMPARE_MATCH_EXAMPLE_VALUE,
```

```
TIMER_COMPARE_MATCH_A);
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
}
```

### Data Structures

| | |
|---|---|
| struct | gpt_output_pin_t |
| struct | gpt_gtior_setting_t |
| struct | gpt_instance_ctrl_t |
| struct | gpt_extended_pwm_cfg_t |
| struct | gpt_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | gpt_io_pin_t |
| enum | gpt_buffer_force_push |
| enum | gpt_pin_level_t |
| enum | gpt_source_t |
| enum | gpt_capture_filter_t |
| enum | gpt_adc_trigger_t |
| enum | gpt_poeg_link_t |
| enum | gpt_output_disable_t |
| enum | gpt_gtioc_disable_t |
| enum | gpt_adc_compare_match_t |
| enum | gpt_interrupt_skip_source_t |
| enum | gpt_interrupt_skip_count_t |
| enum | gpt_interrupt_skip_adc_t |

| | |
|---|---|
| enum | gpt_pwm_output_delay_setting_t |
| enum | gpt_pwm_output_delay_edge_t |
| enum | gpt_gtioc_polarity_t |

## Data Structure Documentation

### ◆ gpt_output_pin_t

| struct gpt_output_pin_t | | |
|---|---|---|
| Configurations for output pins. | | |
| Data Fields | | |
| bool | output_enabled | Set to true to enable output, false to disable output. |
| gpt_pin_level_t | stop_level | Select a stop level from gpt_pin_level_t. |

### ◆ gpt_gtior_setting_t

| struct gpt_gtior_setting_t |
|---|
| Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins. |

### ◆ gpt_instance_ctrl_t

| struct gpt_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

### ◆ gpt_extended_pwm_cfg_t

| struct gpt_extended_pwm_cfg_t | | |
|---|---|---|
| GPT extension for advanced PWM features. | | |
| Data Fields | | |
| uint8_t | trough_ipl | Trough interrupt priority. |
| IRQn_Type | trough_irq | Trough interrupt. |
| gpt_poeg_link_t | poeg_link | Select which POEG channel controls output disable for this GPT channel. |
| gpt_output_disable_t | output_disable | Select which trigger sources request output disable from POEG. |
| gpt_adc_trigger_t | adc_trigger | Select trigger sources to start A/D conversion. |

| | | |
|---|---|---|
| uint32_t | dead_time_count_up | Set a dead time value for counting up. |
| uint32_t | dead_time_count_down | Set a dead time value for counting down (available on GPT32E and GPT32EH only) |
| uint32_t | adc_a_compare_match | Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_A. |
| uint32_t | adc_b_compare_match | Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_B. |
| gpt_interrupt_skip_source_t | interrupt_skip_source | Interrupt source to count for interrupt skipping. |
| gpt_interrupt_skip_count_t | interrupt_skip_count | Number of interrupts to skip between events. |
| gpt_interrupt_skip_adc_t | interrupt_skip_adc | ADC events to skip when interrupt skipping is enabled. |
| gpt_gtioc_disable_t | gtioca_disable_setting | Select how to configure GTIOCA when output is disabled. |
| gpt_gtioc_disable_t | gtiocb_disable_setting | Select how to configure GTIOCB when output is disabled. |

### ◆ gpt_extended_cfg_t

| struct gpt_extended_cfg_t | | |
|---|---|---|
| GPT extension configures the output pins for GPT. | | |
| Data Fields | | |
| gpt_output_pin_t | gtioca | Configuration for GPT I/O pin A. |
| gpt_output_pin_t | gtiocb | Configuration for GPT I/O pin B. |
| gpt_source_t | start_source | Event sources that trigger the timer to start. |
| gpt_source_t | stop_source | Event sources that trigger the timer to stop. |
| gpt_source_t | clear_source | Event sources that trigger the timer to clear. |
| gpt_source_t | capture_a_source | Event sources that trigger capture of GTIOCA. |
| gpt_source_t | capture_b_source | Event sources that trigger capture of GTIOCB. |
| gpt_source_t | count_up_source | Event sources that trigger a |

| | | |
|---|---|---|
| | | single up count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK. |
| gpt_source_t | count_down_source | Event sources that trigger a single down count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK. |
| gpt_capture_filter_t | capture_filter_gtioca | |
| gpt_capture_filter_t | capture_filter_gtiocb | |
| uint8_t | capture_a_ipl | Capture A interrupt priority. |
| uint8_t | capture_b_ipl | Capture B interrupt priority. |
| IRQn_Type | capture_a_irq | Capture A interrupt. |
| IRQn_Type | capture_b_irq | Capture B interrupt. |
| uint32_t | compare_match_value[2] | Storing compare match value for channels. |
| uint8_t | compare_match_status | Storing the compare match register status. |
| gpt_extended_pwm_cfg_t const * | p_pwm_cfg | Advanced PWM features, optional. |
| gpt_gtior_setting_t | gtior_setting | Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins. |
| gpt_gtioc_polarity_t | gtioca_polarity | Polarity control for GTIOCxA input/output pin. |
| gpt_gtioc_polarity_t | gtiocb_polarity | Polarity control for GTIOCxB input/output pin. |

### Enumeration Type Documentation

### ◆ gpt_io_pin_t

| enum gpt_io_pin_t | |
|---|---|
| Input/Output pins, used to select which duty cycle to update in R_GPT_DutyCycleSet(). | |
| Enumerator | |
| GPT_IO_PIN_GTIOCA | GTIOCA. |
| GPT_IO_PIN_GTIOCB | GTIOCB. |
| GPT_IO_PIN_GTIOCA_AND_GTIOCB | GTIOCA and GTIOCB. |
| GPT_IO_PIN_TROUGH | Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected. |
| GPT_IO_PIN_CREST | Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected. |
| GPT_IO_PIN_ONE_SHOT_LEADING_EDGE | Used in R_GPT_DutyCycleSet to set GTCCRC and GTCCRE registers when One-Shot Pulse mode is selected. |
| GPT_IO_PIN_ONE_SHOT_TRAILING_EDGE | Used in R_GPT_DutyCycleSet to set GTCCRD and GTCCRF registers when One-Shot Pulse mode is selected. |

### ◆ gpt_buffer_force_push

| enum gpt_buffer_force_push | |
|---|---|
| Forced buffer push operation used in One-Sot Pulse mode with R_GPT_DutyCycleSet(). | |
| Enumerator | |
| GPT_BUFFER_FORCE_PUSH | Used in R_GPT_DutyCycleSet to force push the data from GTCCRn registers to temporary buffer A or B when One-Shot Pulse mode is selected. |

### ◆ gpt_pin_level_t

| enum gpt_pin_level_t | |
|---|---|
| Level of GPT pin | |
| Enumerator | |
| GPT_PIN_LEVEL_LOW | Pin level low. |
| GPT_PIN_LEVEL_HIGH | Pin level high. |

### ◆ gpt_source_t

| enum gpt_source_t | |
|---|---|
| Sources can be used to start the timer, stop the timer, count up, or count down. These enumerations represent a bitmask. Multiple sources can be ORed together. | |
| Enumerator | |
| GPT_SOURCE_NONE | No active event sources. |
| GPT_SOURCE_GTETRGA_RISING | Action performed on GTETRGA rising edge. |
| GPT_SOURCE_GTETRGA_FALLING | Action performed on GTETRGA falling edge. |
| GPT_SOURCE_GTETRGB_RISING | Action performed on GTETRGB rising edge. |
| GPT_SOURCE_GTETRGB_FALLING | Action performed on GTETRGB falling edge. |
| GPT_SOURCE_GTETRGC_RISING | Action performed on GTETRGC rising edge. |
| GPT_SOURCE_GTETRGC_FALLING | Action performed on GTETRGC falling edge. |
| GPT_SOURCE_GTETRGD_RISING | Action performed on GTETRGB rising edge. |
| GPT_SOURCE_GTETRGD_FALLING | Action performed on GTETRGB falling edge. |
| GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_LOW | Action performed when GTIOCA input rises while GTIOCB is low. |
| GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_HIGH | Action performed when GTIOCA input rises while GTIOCB is high. |
| GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_LOW | Action performed when GTIOCA input falls while GTIOCB is low. |
| GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_HIGH | Action performed when GTIOCA input falls while GTIOCB is high. |

| | |
|---|---|
| GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_LOW | Action performed when GTIOCB input rises while GTIOCA is low. |
| GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_HIGH | Action performed when GTIOCB input rises while GTIOCA is high. |
| GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_LOW | Action performed when GTIOCB input falls while GTIOCA is low. |
| GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_HIGH | Action performed when GTIOCB input falls while GTIOCA is high. |
| GPT_SOURCE_GPT_A | Action performed on ELC GPTA event. |
| GPT_SOURCE_GPT_B | Action performed on ELC GPTB event. |
| GPT_SOURCE_GPT_C | Action performed on ELC GPTC event. |
| GPT_SOURCE_GPT_D | Action performed on ELC GPTD event. |
| GPT_SOURCE_GPT_E | Action performed on ELC GPTE event. |
| GPT_SOURCE_GPT_F | Action performed on ELC GPTF event. |
| GPT_SOURCE_GPT_G | Action performed on ELC GPTG event. |
| GPT_SOURCE_GPT_H | Action performed on ELC GPTH event. |

### ◆ gpt_capture_filter_t

enum gpt_capture_filter_t

Input capture signal noise filter (debounce) setting. Only available for input signals GTIOCxA and GTIOCxB. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, GPT section.

| Enumerator | |
|---|---|
| GPT_CAPTURE_FILTER_NONE | None - no filtering. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_1 | PCLK/1 - fast sampling. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_4 | PCLK/4. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_16 | PCLK/16. |
| GPT_CAPTURE_FILTER_PCLKD_DIV_64 | PCLK/64 - slow sampling. |

### ◆ gpt_adc_trigger_t

| enum gpt_adc_trigger_t | |
|---|---|
| Trigger options to start A/D conversion. | |
| Enumerator | |
| GPT_ADC_TRIGGER_NONE | None - no output disable request. |
| GPT_ADC_TRIGGER_UP_COUNT_START_ADC_A | Request A/D conversion from ADC unit 0 at up counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match. |
| GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_A | Request A/D conversion from ADC unit 0 at down counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match. |
| GPT_ADC_TRIGGER_UP_COUNT_START_ADC_B | Request A/D conversion from ADC unit 1 at up counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match. |
| GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_B | Request A/D conversion from ADC unit 1 at down counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match. |

### ◆ gpt_poeg_link_t

| enum gpt_poeg_link_t | |
|---|---|
| POEG channel to link to this channel. | |
| Enumerator | |
| GPT_POEG_LINK_POEG0 | Link this GPT channel to POEG channel 0 (GTETRGA) |
| GPT_POEG_LINK_POEG1 | Link this GPT channel to POEG channel 1 (GTETRGB) |
| GPT_POEG_LINK_POEG2 | Link this GPT channel to POEG channel 2 (GTETRGC) |
| GPT_POEG_LINK_POEG3 | Link this GPT channel to POEG channel 3 (GTETRGD) |

### ◆ gpt_output_disable_t

| enum gpt_output_disable_t | |
|---|---|
| Select trigger to send output disable request to POEG. | |
| Enumerator | |
| GPT_OUTPUT_DISABLE_NONE | None - no output disable request. |
| GPT_OUTPUT_DISABLE_DEAD_TIME_ERROR | Request output disable if a dead time error occurs. |
| GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_HIGH | Request output disable if GTIOCA and GTIOCB are high at the same time. |
| GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_LOW | Request output disable if GTIOCA and GTIOCB are low at the same time. |

### ◆ gpt_gtioc_disable_t

| enum gpt_gtioc_disable_t | |
|---|---|
| Disable level options for GTIOC pins. | |
| Enumerator | |
| GPT_GTIOC_DISABLE_PROHIBITED | Do not allow output disable. |
| GPT_GTIOC_DISABLE_SET_HI_Z | Set GTIOC to high impedance when output is disabled. |
| GPT_GTIOC_DISABLE_LEVEL_LOW | Set GTIOC level low when output is disabled. |
| GPT_GTIOC_DISABLE_LEVEL_HIGH | Set GTIOC level high when output is disabled. |

### ◆ gpt_adc_compare_match_t

| enum gpt_adc_compare_match_t | |
|---|---|
| Trigger options to start A/D conversion. | |
| Enumerator | |
| GPT_ADC_COMPARE_MATCH_ADC_A | Set A/D conversion start request value for GPT A/D converter start request A. |
| GPT_ADC_COMPARE_MATCH_ADC_B | Set A/D conversion start request value for GPT A/D converter start request B. |

### ◆ gpt_interrupt_skip_source_t

| enum gpt_interrupt_skip_source_t | |
|---|---|
| Interrupt skipping modes | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_SOURCE_NONE | Do not skip interrupts. |
| GPT_INTERRUPT_SKIP_SOURCE_OVERFLOW_UNDERFLOW | Count and skip overflow and underflow interrupts. |
| GPT_INTERRUPT_SKIP_SOURCE_CREST | Count crest interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the trough interrupt fires before the crest interrupt. |
| GPT_INTERRUPT_SKIP_SOURCE_TROUGH | Count trough interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the crest interrupt fires before the trough interrupt. |

### ◆ gpt_interrupt_skip_count_t

| enum gpt_interrupt_skip_count_t | |
|---|---|
| Number of interrupts to skip between events | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_COUNT_0 | Do not skip interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_1 | Skip one interrupt. |
| GPT_INTERRUPT_SKIP_COUNT_2 | Skip two interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_3 | Skip three interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_4 | Skip four interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_5 | Skip five interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_6 | Skip six interrupts. |
| GPT_INTERRUPT_SKIP_COUNT_7 | Skip seven interrupts. |

### ◆ gpt_interrupt_skip_adc_t

| enum gpt_interrupt_skip_adc_t | |
|---|---|
| ADC events to skip during interrupt skipping | |
| Enumerator | |
| GPT_INTERRUPT_SKIP_ADC_NONE | Do not skip ADC events. |
| GPT_INTERRUPT_SKIP_ADC_A | Skip ADC A events. |
| GPT_INTERRUPT_SKIP_ADC_B | Skip ADC B events. |
| GPT_INTERRUPT_SKIP_ADC_A_AND_B | Skip ADC A and B events. |

### ◆ gpt_pwm_output_delay_setting_t

| enum gpt_pwm_output_delay_setting_t | |
|---|---|
| Delay setting for the PWM Delay Generation Circuit (PDG). | |
| Enumerator | |
| GPT_PWM_OUTPUT_DELAY_SETTING_0_32 | Delay is not applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_1_32 | Delay of 1 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_2_32 | Delay of 2 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_3_32 | Delay of 3 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_4_32 | Delay of 4 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_5_32 | Delay of 5 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_6_32 | Delay of 6 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_7_32 | Delay of 7 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_8_32 | Delay of 8 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_9_32 | Delay of 9 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_10_32 | Delay of 10 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_11_32 | Delay of 11 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_12_32 | Delay of 12 / 32 GTCLK period applied. |

| | |
|---|---|
| GPT_PWM_OUTPUT_DELAY_SETTING_13_32 | Delay of 13 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_14_32 | Delay of 14 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_15_32 | Delay of 15 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_16_32 | Delay of 16 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_17_32 | Delay of 17 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_18_32 | Delay of 18 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_19_32 | Delay of 19 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_20_32 | Delay of 20 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_21_32 | Delay of 21 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_22_32 | Delay of 22 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_23_32 | Delay of 23 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_24_32 | Delay of 24 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_25_32 | Delay of 25 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_26_32 | Delay of 26 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_27_32 | Delay of 27 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_28_32 | Delay of 28 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_29_32 | Delay of 29 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_30_32 | Delay of 30 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_31_32 | Delay of 31 / 32 GTCLK period applied. |
| GPT_PWM_OUTPUT_DELAY_SETTING_BYPASS | Bypass the PWM Output Delay Circuit. |

### ◆ gpt_pwm_output_delay_edge_t

| enum gpt_pwm_output_delay_edge_t | |
|---|---|
| Select which PWM Output Delay setting to apply. | |
| Enumerator | |
| GPT_PWM_OUTPUT_DELAY_EDGE_RISING | Configure the PWM Output Delay setting for rising edge. |
| GPT_PWM_OUTPUT_DELAY_EDGE_FALLING | Configure the PWM Output Delay setting for falling edge. |

### ◆ gpt_gtioc_polarity_t

| enum gpt_gtioc_polarity_t | |
|---|---|
| Polarity inversion control for GTIOCnA/B pins. | |
| Enumerator | |
| GPT_GTIOC_POLARITY_NORMAL | GPTIOCnA/B pin polarity is not changed. |
| GPT_GTIOC_POLARITY_INVERTED | GPTIOCnA/B pin polarity is inverted. |

## Function Documentation

### ◆ R_GPT_Open()

fsp_err_t R_GPT_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the timer module and applies configurations. Implements timer_api_t::open.

GPT hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The GPT implementation of the general timer can accept a gpt_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the source divider is invalid. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback. |
| FSP_ERR_INVALID_MODE | Triangle wave PWM is only supported if GPT_CFG_OUTPUT_SUPPORT_ENABLE is 2. Selected channel does not support external count sources. External and event count sources not are available in this mode. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The channel requested in the p_cfg parameter is not available on this device. |

### ◆ R_GPT_Stop()

fsp_err_t R_GPT_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_GPT_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Start()

fsp_err_t R_GPT_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Reset()

fsp_err_t R_GPT_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

***Note***

*This function also updates to the new period if no counter overflow has occurred since the last call to R_GPT_PeriodSet().*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Enable()

fsp_err_t R_GPT_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_GPT_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_Disable()

fsp_err_t R_GPT_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

***Note***

*The timer could be running after R_GPT_Disable(). To ensure it is stopped, call R_GPT_Stop().*

Example:

```
/* (Optional) Disable captures. */
   (void) R_GPT_Disable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_PeriodSet()

fsp_err_t R_GPT_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Sets period value provided. If the timer is running, the period will be updated after the next counter overflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and a GPT overflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter overflow after processing completes.

Example:

```
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
```

```
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_DutyCycleSet()

fsp_err_t R_GPT_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Duty cycle is updated in the buffer register. The updated duty cycle is reflected after the next cycle end (counter overflow).

Example:

```
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
```

```
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block. |
| [in] | duty_cycle_counts | Duty cycle to set in counts. |
| [in] | pin | Use gpt_io_pin_t to select GPT_IO_PIN_GTIOCA or GPT_IO_PIN_GTIOCB |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL or the pin is not one of gpt_io_pin_t |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle is larger than period. |
| FSP_ERR_INVALID_MODE | GPT_IO_PIN_TROUGH, and GPT_IO_PIN_CREST settings are invalid in the this mode. |
| FSP_ERR_UNSUPPORTED | GPT_CFG_OUTPUT_SUPPORT_ENABLE is 0. |

### ◆ R_GPT_CompareMatchSet()

fsp_err_t R_GPT_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

Set value for compare match feature. Implements timer_api_t::compareMatchSet.

***Note***

*This API should be used when timer is stop counting. And shall not be used along with PWM operation.*

Example:

```
 /* Set the compare match value (GPT_COMPARE_MATCH_EXAMPLE_VALUE). This value must be
less than or equal to period value. */
    err = R_GPT_CompareMatchSet(&g_timer0_ctrl, GPT_COMPARE_MATCH_EXAMPLE_VALUE,
TIMER_COMPARE_MATCH_A);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Set the compare match value successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_NOT_ENABLED | Requested compare channel is disabled. |

### ◆ R_GPT_InfoGet()

fsp_err_t R_GPT_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
      uint64_t period = info.period_counts;
/* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
 * timer_info_t::period_counts. */
if (0U == period)
    {
          period = UINT32_MAX + 1U;
    }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, frequency, and ELC event written to caller's structure successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_info was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_StatusGet()

fsp_err_t R_GPT_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current timer state and counter value set successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_CounterSet()

fsp_err_t R_GPT_CounterSet ( timer_ctrl_t *const *p_ctrl*, uint32_t *counter* )

Set counter value.

***Note***

*Do not call this API while the counter is counting. The counter value can only be updated while the counter is stopped.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value updated. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_IN_USE | The timer is running. Stop the timer before calling this function. |

### ◆ R_GPT_OutputEnable()

fsp_err_t R_GPT_OutputEnable ( timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin* )

Enable output for GTIOCA and/or GTIOCB.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Output is enabled. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_OutputDisable()

fsp_err_t R_GPT_OutputDisable ( timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin* )

Disable output for GTIOCA and/or GTIOCB.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Output is disabled. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_AdcTriggerSet()

fsp_err_t R_GPT_AdcTriggerSet ( timer_ctrl_t *const *p_ctrl*, gpt_adc_compare_match_t *which_compare_match*, uint32_t *compare_match_value* )

Set A/D converter start request compare match value.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value updated. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_PwmOutputDelaySet()

fsp_err_t R_GPT_PwmOutputDelaySet ( timer_ctrl_t *const *p_ctrl*, gpt_pwm_output_delay_edge_t *edge*, gpt_pwm_output_delay_setting_t *delay_setting*, uint32_t const *pin* )

Set the Output Delay setting for the PWM output pin.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | The output delay was set. |
| FSP_ERR_ASSERTION | An input parameter was invalid. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_CHANNEL | The channel does not support this feature. |
| FSP_ERR_NOT_INITIALIZED | The PWM Output Delay Circuit has not been initialized. |
| FSP_ERR_INVALID_STATE | The PWM Output Delay setting cannot be updated in the current state. |
| FSP_ERR_UNSUPPORTED | This feature is not supported on this MCU. |

### ◆ R_GPT_CallbackSet()

fsp_err_t R_GPT_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_GPT_Close()

fsp_err_t R_GPT_Close ( timer_ctrl_t *const *p_ctrl*)

Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_GPT_PwmOutputDelayInitialize()

fsp_err_t R_GPT_PwmOutputDelayInitialize ( )

Initialize the PWM Delay Generation Circuit (PDG). This function must be called before calling R_GPT_PwmOutputDelaySet.

***Note***

*This function will delay for 20 microseconds.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization sequence completed successfully. |
| FSP_ERR_INVALID_STATE | The source clock frequnecy is out of the required range for the PDG. |
| FSP_ERR_UNSUPPORTED | This feature is not supported. |

## 5.2.19.8 Timer, Low-Power (r_agt)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_AGT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_AGT_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_Enable (timer_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | R_AGT_Disable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_AGT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_AGT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_AGT_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_AGT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_AGT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_AGT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_AGT_Close (timer_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the AGT and AGTW peripheral on RA MCUs. This module implements the Timer Interface.

# Overview

### Features

The AGT module has the following features:

- Supports 16- and 32-bit timers via the AGT and AGTW peripherals, respectively.
- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, including PCLKB, LOCO, SUBCLK, and external sources input to AGTIO.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports counting based on an external clock input to AGTIO.
- Supports debounce filter on AGTIO pins.
- Supports measuring pulse width or pulse period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.

### Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

| | GPT | AGT |
|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The AGT can operate in all low power modes (when count source is LOCO or subclock). |
| Available Channels | The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels. | All MCUs have at least 2 AGT channels. |
| Timer Resolution | All MCUs have at least one 32-bit GPT timer. | The AGT timers can be 16-bit or 32-bit timers. |
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The AGT runs off PCLKB, LOCO, or subclock with a configurable divider up to 8 for PCLKB or up to 128 for LOCO or subclock. |

# Configuration

### Build Time Configurations for r_agt

The following build time configurations are defined in fsp_cfg/r_agt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Pin Output Support | • Disabled<br>• Enabled | Disabled | If selected code for outputting a waveform to a pin is included in the build. |
| Pin Input Support | • Disabled<br>• Enabled | Disabled | Enable input support to use pulse width measurement mode, pulse period measurement mode, or input from P402, P403, P404, or AGTIO. |

### Configurations for Timers > Timer, Low-Power (r_agt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, Low-Power (r_agt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Counter Bit Width | MCU Specific Options | | Counter register bit width (16-bit or 32-bit)) |
| Channel | Channel number must be a non-negative integer | 0 | Physical hardware channel. |
| Mode | • Periodic<br>• One-Shot<br>• PWM | Periodic | Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused. |
| Period | Value must be non-negative | 0x10000 | Specify the timer period based on the selected unit.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x100000000 (32-bit) or 0x10000 (16-bit) results in the maximum period at the lowest divisor (fastest timer tick). Set the period to 0x100000000 (32-bit) or 0x10000 (16-bit) for a free running timer, pulse width measurement or pulse period measurement. Setting the period higher will automatically select a higher divider; the period can be set up to 0x800000000 (32-bit) or 0x80000 (16-bit) when counting from PCLKB or 0x8000000000 (32-bit) or 0x800000 (16-bit) when counting from LOCO/subclock, which will use a divider of 8 or 128 respectively with the maximum |

| | | | |
|---|---|---|---|
| | | | period.<br><br>If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the timer_cfg_t structure. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Count Source | MCU Specific Options | | AGT counter clock source. NOTE: The divisor is calculated automatically based on the selected period. See agt_clock_t documentation for details. |
| Output | | | |
| Duty Cycle Percent (only applicable in PWM mode) | Value must be between 0 and 100 | 50 | Specify the timer duty cycle percent. Only used in PWM mode. |
| AGTOA Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTOA output. |
| AGTOB Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTOB output. |
| AGTO Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure AGTO output. |
| Input | | | |
| Measurement Mode | • Measure Disabled<br>• Measure Low Level Pulse Width<br>• Measure High Level Pulse | Measure Disabled | Select if the AGT should be used to measure pulse width or pulse period. In high level pulse width measurement mode, the AGT counts when |

| | | | |
|---|---|---|---|
| | Width<br>• Measure Pulse Period | | AGTIO is high and starts counting immediately in the middle of a pulse if AGTIO is high when R_AGT_Start() is called. In low level pulse width measurement mode, the AGT counts when AGTIO is low and could start counting in the middle of a pulse if AGTIO is low when R_AGT_Start() is called. |
| Input Filter | • No Filter<br>• Filter sampled at PCLKB<br>• Filter sampled at PCLKB / 8<br>• Filter sampled at PCLKB / 32 | No Filter | Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. |
| Enable Pin | • Enable Pin Not Used<br>• Enable Pin Active Low<br>• Enable Pin Active High | Enable Pin Not Used | Select active edge for the AGTEE pin if used. Only applies if the count source is P402, P403 or AGTIO. |
| Trigger Edge | • Trigger Edge Rising<br>• Trigger Edge Falling<br>• Trigger Edge Both | Trigger Edge Rising | Select the trigger edge. Applies if measurement mode is pulse period, or if the count source is P402, P403, or AGTIO. Do not select Trigger Edge Both with pulse period measurement. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses. |
| Underflow Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

When using the AGT module on devices with both AGT and AGTW peripherals, the channel numbers used with the AGT driver are combined between both peripherals; AGTW channels are listed first followed by AGT. For example, on the RA2A2 there are 2 AGTW channels and 8 AGT channels. These correspond to channels 0-1 for AGTW0-AGTW1 and 2-9 for AGT0-AGT7.

### Clock Configuration

The AGT clock is based on the PCLKB, LOCO, or Subclock frequency. You can set the clock frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module can use the AGTOA and AGTOB pins as output pins for periodic, one-shot, or PWM signals.

For input capture, the input signal must be applied to the AGTIOn pin.

For event counting, the AGTEEn enable pin is optional.

### Timer Period

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units, and clock speed.

When the selected unit is "Raw counts", the maximum allowed period setting varies depending on the selected clock source:

| Clock source | 16-bit Timer Maximum period (counts) | 32-bit Timer Maximum period (counts) |
|---|---|---|
| LOCO/Subclock | 0x800000 | 0x8000000000 |
| PCLKB | 0x80000 | 0x800000000 |
| All other sources | 0x10000 | 0x100000000 |

*Note*

*The period interrupt occurs when the counter underflows, setting the period register to 0 results in an effective period of 1 count. For this reason all user-provided raw count values reflect the actual number of period counts (not the raw register values).*
*When using a 32-bit channel (AGTW), a special period count value of 0 is used to obtain an actual period register value of 0xFFFFFFFF.*

# Usage Notes

### Starting and Stopping the AGT

After starting or stopping the timer, AGT registers cannot be accessed until the AGT state is updated after 3 AGTCLK cycles. If another AGT function is called before the 3 AGTCLK period elapses, the function spins waiting for the AGT state to update. The required wait time after starting or stopping the timer can be determined using the frequency of AGTCLK, which is derived from timer_cfg_t::source_div and agt_extended_cfg_t::count_source.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

### Low Power Modes

The AGT1 (channel 1 only) can be used to enter snooze mode or to wake the MCU from snooze, software standby, or deep software standby modes when a counter underflow occurs. The compare match A and B events can also be used to wake from software standby or snooze modes.

### One-Shot Mode

The AGT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Transfer (r_dtc).

### One-Shot Mode Output

The output waveform in one-shot mode is one AGT clock cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

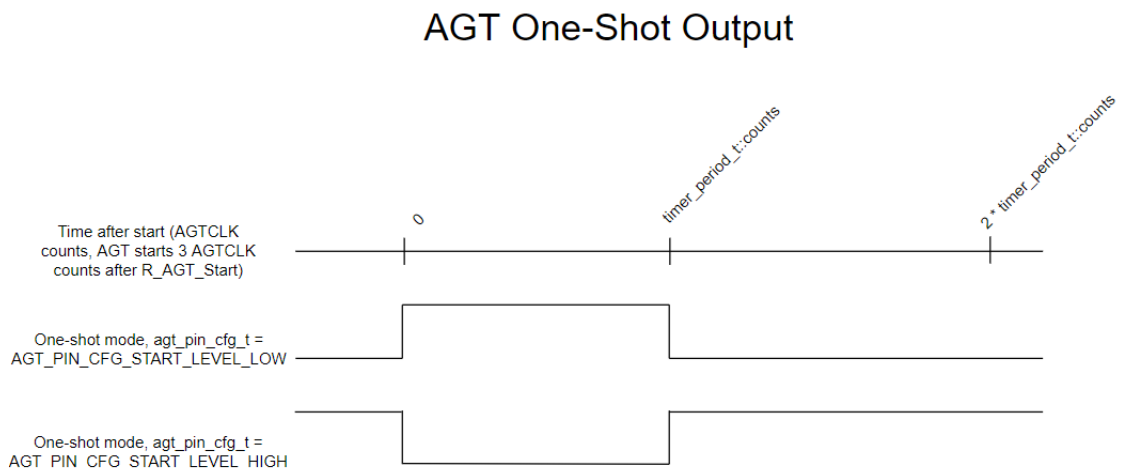



Figure 343: AGT One-Shot Output

### Periodic Output

The AGTOA or AGTOB pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square (from rising edge to rising edge) matches the period of the AGT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

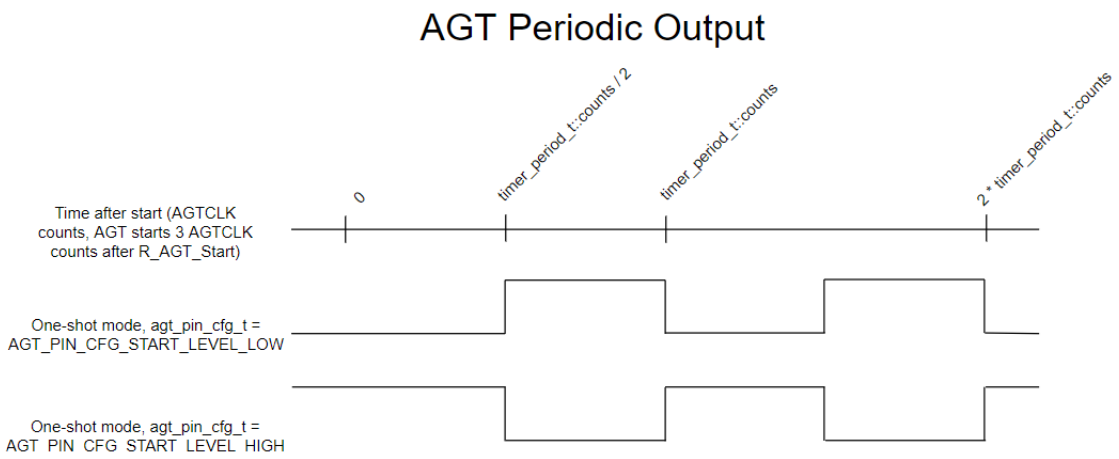



Figure 344: AGT Periodic Output

### PWM Output

This module does not support in phase PWM output. The PWM output signal is low at the beginning of the cycle and high at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

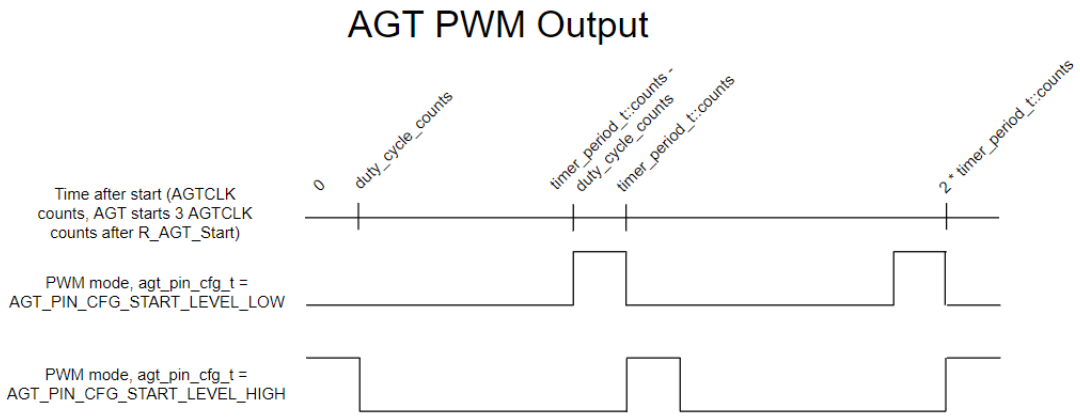



Figure 345: AGT PWM Output

### Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

# Examples

### AGT Basic Example

This is a basic example of minimal use of the AGT in an application.

```
void agt_basic_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
}
```

### AGT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### AGT Free Running Counter Example

To use the AGT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF for a 32-bit timer or 0xFFFF for a 16-bit timer.

```
void agt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_AGT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
}
```

### AGT Input Capture Example

This is an example of using the AGT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CAPTURE_A == p_args->event)
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
       uint32_t period = info.period_counts;
 /* Process capture from AGTIO. */
       g_captured_time    = ((uint64_t) period * g_capture_overflows) +
p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
    }
```

```
}
void agt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_AGT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_AGT_Disable(&g_timer0_ctrl);
}
```

### AGT Period Update Example

This an example of updating the period.

```
#define AGT_EXAMPLE_MSEC_PER_SEC (1000)
#define AGT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void agt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
```

```
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

### AGT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define AGT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void agt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. */
    uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                 AGT_EXAMPLE_MAX_PERCENT;
 /* Set the calculated duty cycle. */
    err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
    assert(FSP_SUCCESS == err);
}
```

### AGT Cascaded Timers Example

This an example of using underflow from an even AGT channel as the count source for the next channel (in this case, AGT0 and AGT1).

```
/* This example shows how use cascaded timers. The count source for AGT channel 1 is
set to AGT0 underflow. */
void agt_cascaded_timers_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the timers in any order. */
    err = R_AGT_Open(&g_timer_channel0_ctrl, &g_timer_channel0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_AGT_Open(&g_timer_channel1_ctrl, &g_timer_channel1_cfg);
    assert(FSP_SUCCESS == err);
 /* Start AGT channel 1 first. */
```

```
    (void) R_AGT_Start(&g_timer_channel1_ctrl);
    (void) R_AGT_Start(&g_timer_channel0_ctrl);
 /* (Optional) Stop AGT channel 0 first. */
    (void) R_AGT_Stop(&g_timer_channel0_ctrl);
    (void) R_AGT_Stop(&g_timer_channel1_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer_channel1_ctrl, &status);
}
```

### Data Structures

| | |
|---|---|
| struct | agt_instance_ctrl_t |
| struct | agt_extended_cfg_t |

### Enumerations

| | |
|---|---|
| enum | agt_clock_t |
| enum | agt_measure_t |
| enum | agt_agtio_filter_t |
| enum | agt_enable_pin_t |
| enum | agt_trigger_edge_t |
| enum | agt_output_pin_t |
| enum | agt_pin_cfg_t |
| enum | agt_counter_bit_width_t |

## Data Structure Documentation

### ◆ agt_instance_ctrl_t

| struct agt_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

### ◆ agt_extended_cfg_t

| struct agt_extended_cfg_t |
|---|
| Optional AGT extension data structure. |

| Data Fields | | |
|---|---|---|
| agt_clock_t | count_source | AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_CLOCK_FSUB. |
| union agt_extended_cfg_t | __unnamed__ | |
| agt_pin_cfg_t | agto: 3 | Configure AGTO pin.<br>*Note*<br>*AGTIO polarity is opposite AGTO* |
| agt_measure_t | measurement_mode | Measurement mode. |
| agt_agtio_filter_t | agtio_filter | Input filter for AGTIO. |
| agt_enable_pin_t | enable_pin | Enable pin (event counting only) |
| agt_trigger_edge_t | trigger_edge | Trigger edge to start pulse period measurement or count external event. |
| agt_counter_bit_width_t | counter_bit_width | Counter bit width. |

### Enumeration Type Documentation

### ◆ agt_clock_t

| enum agt_clock_t | |
|---|---|
| Count source | |
| Enumerator | |
| AGT_CLOCK_PCLKB | PCLKB count source, division by 1, 2, or 8 allowed. |
| AGT_CLOCK_LOCO | LOCO count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed. |
| AGT_CLOCK_AGT_UNDERFLOW | Underflow event signal from next lowest AGT channel, division must be 1. |
| AGT_CLOCK_SUBCLOCK | Subclock count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed. |
| AGT_CLOCK_P402 | Counts events on P402, events are counted in deep software standby mode. |
| AGT_CLOCK_P403 | Counts events on P403, events are counted in deep software standby mode. |
| AGT_CLOCK_P404 | Counts events on P404, events are counted in deep software standby mode. |
| AGT_CLOCK_AGTIO | Counts events on AGTIOn, events are not counted in software standby modes. |

### ◆ agt_measure_t

| enum agt_measure_t | |
|---|---|
| Enable pin for event counting mode. | |
| Enumerator | |
| AGT_MEASURE_DISABLED | AGT used as a counter. |
| AGT_MEASURE_PULSE_WIDTH_LOW_LEVEL | AGT used to measure low level pulse width. |
| AGT_MEASURE_PULSE_WIDTH_HIGH_LEVEL | AGT used to measure high level pulse width. |
| AGT_MEASURE_PULSE_PERIOD | AGT used to measure pulse period. |

### ◆ agt_agtio_filter_t

| enum agt_agtio_filter_t | |
|---|---|
| Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. | |
| Enumerator | |
| AGT_AGTIO_FILTER_NONE | No filter. |
| AGT_AGTIO_FILTER_PCLKB | Filter at PCLKB. |
| AGT_AGTIO_FILTER_PCLKB_DIV_8 | Filter at PCLKB / 8. |
| AGT_AGTIO_FILTER_PCLKB_DIV_32 | Filter at PCLKB / 32. |

### ◆ agt_enable_pin_t

| enum agt_enable_pin_t | |
|---|---|
| Enable pin for event counting mode. | |
| Enumerator | |
| AGT_ENABLE_PIN_NOT_USED | AGTEE/AGTWEE is not used. |
| AGT_ENABLE_PIN_ACTIVE_LOW | Events are only counted when AGTEE/AGTWEE is low. |
| AGT_ENABLE_PIN_ACTIVE_HIGH | Events are only counted when AGTEE/AGTWEE is high. |

### ◆ agt_trigger_edge_t

| enum agt_trigger_edge_t | |
|---|---|
| Trigger edge for pulse period measurement mode and event counting mode. | |
| Enumerator | |
| AGT_TRIGGER_EDGE_RISING | Measurement starts or events are counted on rising edge. |
| AGT_TRIGGER_EDGE_FALLING | Measurement starts or events are counted on falling edge. |
| AGT_TRIGGER_EDGE_BOTH | Events are counted on both edges (n/a for pulse period mode) |

### ◆ agt_output_pin_t

| enum agt_output_pin_t | |
|---|---|
| Output pins, used to select which duty cycle to update in R_AGT_DutyCycleSet(). | |
| Enumerator | |
| AGT_OUTPUT_PIN_AGTOA | AGTOA. |
| AGT_OUTPUT_PIN_AGTOB | AGTOB. |

### ◆ agt_pin_cfg_t

| enum agt_pin_cfg_t | |
|---|---|
| Level of AGT pin | |
| Enumerator | |
| AGT_PIN_CFG_DISABLED | Not used as output pin. |
| AGT_PIN_CFG_START_LEVEL_LOW | Pin level low. |
| AGT_PIN_CFG_START_LEVEL_HIGH | Pin level high. |

### ◆ agt_counter_bit_width_t

| enum agt_counter_bit_width_t | |
|---|---|
| Counter type to determine regsiter size | |
| Enumerator | |
| AGT_COUNTER_BIT_WIDTH_DEFAULT | Legacy. |
| AGT_COUNTER_BIT_WIDTH_16 | AGT. |
| AGT_COUNTER_BIT_WIDTH_32 | AGTW. |

## Function Documentation

### ◆ R_AGT_Open()

fsp_err_t R_AGT_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the AGT module instance. Implements timer_api_t::open.

The AGT hardware does not support one-shot functionality natively. The one-shot feature is therefore implemented in the AGT HAL layer. For a timer configured as a one-shot timer, the timer is stopped upon the first timer expiration.

The AGT implementation of the general timer can accept an optional agt_extended_cfg_t extension parameter. For AGT, the extension specifies the clock to be used as timer source and the output pin configurations. If the extension parameter is not specified (NULL), the default clock PCLKB is used and the output pins are disabled.

Example:

```
/* Initializes the module. */
   err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the period is not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_ALREADY_OPEN | R_AGT_Open has already been called for this p_ctrl. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt has not been enabled in the vector table. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel number is not available on AGT. |

### ◆ R_AGT_Start()

fsp_err_t R_AGT_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_AGT_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer started. |
| FSP_ERR_ASSERTION | p_ctrl is null. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Stop()

fsp_err_t R_AGT_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops the timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_AGT_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Reset()

fsp_err_t R_AGT_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the period minus one. Implements timer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter reset. |
| FSP_ERR_ASSERTION | p_ctrl is NULL |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_Enable()

fsp_err_t R_AGT_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_AGT_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_AGT_Disable()

fsp_err_t R_AGT_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Example:

```
/* (Optional) Disable captures. */
   (void) R_AGT_Disable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_AGT_PeriodSet()

fsp_err_t R_AGT_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Updates period. The new period is updated immediately and the counter is reset to the maximum value. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and an AGT underflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter underflow after processing completes.
Stop the timer before calling this function if one-shot output is used.

Example:

```
/* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
 * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
 * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
 * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
 * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
 * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
 * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
 * by timer_cfg_t::source_div.
 *
```

```
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the period was not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_DutyCycleSet()

fsp_err_t R_AGT_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Updates duty cycle. If the timer is counting, the new duty cycle is reflected after the next counter underflow. Implements timer_api_t::dutyCycleSet.

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. */
   uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                AGT_EXAMPLE_MAX_PERCENT;
/* Set the calculated duty cycle. */
   err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the pin was not AGT_AGTO_AGTOA or AGT_AGTO_AGTOB. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle was not in the valid range of 0 to period (counts) - 1 |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |
| FSP_ERR_UNSUPPORTED | AGT_CFG_OUTPUT_SUPPORT_ENABLE is 0. |

### ◆ R_AGT_CompareMatchSet()

fsp_err_t R_AGT_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | AGT compare match is not supported. |

### ◆ R_AGT_InfoGet()

fsp_err_t R_AGT_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
      uint32_t period = info.period_counts;
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, and frequency stored in p_info. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_StatusGet()

fsp_err_t R_AGT_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Retrieves the current state and counter value stores them in p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current status and counter value provided in p_status. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_AGT_CallbackSet()

fsp_err_t R_AGT_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_AGT_Close()

| fsp_err_t R_AGT_Close ( timer_ctrl_t *const *p_ctrl*) |
|---|
| Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Timer closed.<br>FSP_ERR_ASSERTION — p_ctrl is NULL.<br>FSP_ERR_NOT_OPEN — The instance control structure is not opened. |

## 5.2.19.9 Timer, Simultaneous Channel (r_tau_pwm)

Modules » Timers

### Functions

| | |
|---|---|
| fsp_err_t | R_TAU_PWM_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_TAU_PWM_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_PWM_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_PWM_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_PWM_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_TAU_PWM_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_TAU_PWM_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_TAU_PWM_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_TAU_PWM_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_TAU_PWM_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |

| | |
|---|---|
| fsp_err_t | R_TAU_PWM_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_TAU_PWM_Close (timer_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the TAU_PWM peripheral on RA MCUs. This module implements the Timer Interface.

# Overview

### Features

The TAU_PWM module has the following features:

- Supports simultaneous channel operation functions: one-shot pulse output, PWM output and multiple PWM output.
- Configurable clock source (CK00, CK01).
- Build-time configuration of clock divider
- Configurable period (counts per timer cycle).
- Configurable trigger source (TImn input) for one-shot pulse output
- Supports noise filter on input source in one-shot pulse output mode (always enabled)
- Supports runtime reconfiguration of period/delay, pulse width or duty cycle percent.
- Signal can be output to a pin.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Build-time availability of multi-slave mode for PWM outputs

# Configuration

### Build Time Configurations for r_tau_pwm

The following build time configurations are defined in fsp_cfg/r_tau_pwm_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| One-shot Pulse Output Mode Support | • Disabled<br>• Enabled | Disabled | Controls whether one-shot pulse output mode support is included in the build. This setting applies globally to all r_tau_pwm_instances. If one shot mode is not used by any instance, disable this setting to |

| | | | |
|---|---|---|---|
| | | | reduce ROM usage. |
| PWM Output Mode Support | • Disabled<br>• Enabled | Enabled | Controls whether PWM mode support is included in the build. This setting applies globally to all r_tau_pwm_instances. If PWM mode is not used by any instance, disable this setting to reduce ROM usage. |
| Multi-Slave | • Disabled<br>• Enabled | Disabled | Enable support for multiple slaves |

**Configurations for Timers > Timer, Simultaneous Channel Operation (r_tau_pwm)**

This module can be added to the Stacks tab via New Stack > Timers > Timer, Simultaneous Channel Operation (r_tau_pwm).

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Operation Clock | • CK00<br>• CK01 | CK00 | Select the operation clock |
| Mode | • One-shot pulse output<br>• PWM output | PWM output | Mode selection. |
| Master Channel | Channel number must be even. | 0 | Select the TAU master channel. When two or more master channels are to be used, slave channels with a master channel between them may not be set. |
| Period | Value must be a non-negative integer | 0x10000 | Specify the timer period based on the selected unit. In One-shot pulse output mode this value corresponds to the delay time.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x10000/0x10001 results in the maximum period for PWM Output function/One-shot pulse output function |

| | | | |
|---|---|---|---|
| | | | at the lowest divisor (fastest timer tick). The theoretical calculated period is printed in a comment in the timer_cfg_t structure. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Input (One-shot pulse) | | | |
| Trigger Source | MCU Specific Options | | Select the trigger source for master channel. |
| Detect Edge | • Falling Edge<br>• Rising Edge<br>• Both Edges | Falling Edge | Select the detect edge. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function can be provided. If provided, the callback function is called from the interrupt service routine (ISR) each time the timer period elapses in PWM mode, and when the timer delay elapses in one-shot mode. If the optional slave channel interrupt is enabled, the callback is also called when the signal switches state during the PWM cycle in PWM mode, and when the pulse output is complete in one-shot mode. |
| Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

**Configurations for Timers > TAU PWM Channel Configuration (r_tau_pwm)**

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |

| | | | |
|---|---|---|---|
| Channel | Value must be between 0 and 7. | 1 | Specify the slave channel. |
| Output | | | |
| Output Level | • Start Level Low<br>• Start Level High | Start Level Low | Output level of TAU slave channel. |
| Output Polarity | • Active-high<br>• Active-low | Active-high | Output polarity of TAU slave channel. |
| One-Shot Pulse Width | Value must be a non-negative integer | 0xffff | Specify the pulse width based on the selected unit.<br><br>When the unit is set to 'Raw Counts', setting the period to 0xffff results in the maximum period at the lowest divisor (fastest timer tick). |
| One-Shot Pulse Width Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| PWM Duty Cycle Percent | Value must be a non-negative integer between 0 and 100 | 50 | Specify the duty cycle percent for output pulse of slave channel. |
| Interrupts | | | |
| Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

**Clock Configuration**

The TAU PWM clock is based on the peripheral module clock (PCLKB) which is equal to the system clock (ICLK).

Each TAU PWM channel has certain operation clocks selections, these can be set with the **General>Operation Clock** property in the module configuration. When the operation clock of a channel is set to CK00, or CK01, the TAU_PWM module provides divisor values for each of those clocks. These divisors may be set in the **Clocks** tab. As such, setting a divisor in the **Clocks** tab affects all TAU PWM and TAU channels that use that CK0x clock as an input. Adjusting these settings determines the frequency range achievable by a TAU_PWM channel. If a desired frequency is not achievable, the divider in the **Clocks** tab may be adjusted. The clock dividers cannot be adjusted at runtime.

**Pin Configuration**

This module can use the TOmn pins as output pins for One-shot and PWM signals.

This module can use the TImn pin as the input pin for One-shot signals.

### Timer Period And Duty Cycle

The RA Configuration editor will automatically calculate the period/pulse width/duty cycle percent based on the selected period time, units, clock source, clock divider and mode.

When the selected unit is "Raw counts", the maximum and minimum allowed period/pulse width/duty cycle setting are presented by the following tables:

In One-shot pulse output mode:

| Clock divider | Minimum period (counts) | Maximum period (counts) | Minimum pulse width (counts) | Maximum pulse width (counts) |
|---|---|---|---|---|
| PCLK/1 | 0x00003 | 0x10001 | 0x0001 | 0xffff |
| PCLK/2 to PCLK/32768 | 0x00002 | 0x10000 | 0x0000 | 0xffff |

In PWM output mode:

| Clock divider | Minimum period (counts) | Maximum period (counts) | Minimum duty cycle (%) | Maximum duty cycle (%) |
|---|---|---|---|---|
| PCLK/1 | 0x00002 | 0x10001 | 0 | 100 |
| PCLK/2 to PCLK/32768 | 0x00001 | 0x10000 | 0 | 100 |

*Note*

*If PCLK (undivided) is selected as the operation clock (CK00, CK01) and TDR0n is set to 0x0000 (n = 0 to 7), interrupt requests output from timer array units cannot be used. So, when PCLK is undivided, the minimum period must be plus one.*
*Because the period interrupt occurs when the counter underflows, setting the period register to 0 results in an effective period of 1 count. For this reason, all user-provided raw count values reflect the actual number of period counts (not the raw register values).*

## Usage Notes

### Updating Period and Duty Cycle

The period and duty cycle are updated after the next counter underflow after calling R_TAU_PWM_PeriodSet() or R_TAU_PWM_DutyCycleSet().

### One-Shot Pulse Output Mode

By using two channels as a set, a one-shot pulse having any delay pulse width can be generated from the signal input to the TI0n pin.

The delay time is counted by the master channel, and the pulse width is counted by the slave channel.

### PWM Output

Two channels can be used as a set to generate a pulse of any period and duty factor (duty cycle). By extending the PWM function and using multiple slave channels, many PWM waveforms with different

duty values can be output.

When multiple slaves are to be connected to a single master, the build time option for multiple slaves needs to be set.

*Note*

*When two or more master channels are to be used, slave channels with a master channel between them may not be set. For example, if channels 0 and 4 are set as master channels, channels 1 to 3 can be set as the slave channels of master channel 0, channels 5 to 7 cannot be set as the slave channels of master channel 0.*

The period is counted by the master channel, and the duty cycle is counted by the slave channel.

### Controlling TAU_PWM with ELC Events

The TAU_PWM timer can be configured to trigger the timer counter when an ELC event occurs.

*Note*

*Triggering the timer using ELC events is supported only when master channel is set to 0 and mode is set to One-shot pulse output.*
*The event links for the ELC must be configured outside this module.*

### Triggering ELC Events with TAU_PWM

The TAU_PWM timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

*Note*

*Only event signals from channel 00 to 03 are available.*

### Limitations

- None

# Examples

### TAU_PWM Basic Example

This is a basic example of minimal use of the TAU PWM (in PWM Output mode) in an application.

```
void tau_pwm_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
```

```
}
```

### TAU_PWM Callback Example

This is an example of a timer callback.

*Note*

*The callback function always called after the period time (in PWM output mode) or delay time (in One-shot pulse output mode) has expired. Additionally, it can be optionally called after the duty cycle (in PWM output mode) or pulse width (in One-shot pulse output mode) has expired by enabling the slave channel's interrupt.*

```
uint32_t g_timer_master_cycle_end_counter = 0;
uint32_t g_timer_slave_cycle_end_counter = 0;
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_MASTER_CYCLE_END == p_args->event)
    {
       g_timer_master_cycle_end_counter++;
    }
 if (TIMER_EVENT_SLAVE_CYCLE_END == p_args->event)
    {
       g_timer_slave_cycle_end_counter++;
    }
}
```

### TAU_PWM One-Shot Pulse Output Mode Example

### TAU_PWM One-Shot Pulse Output Mode Example - Software Trigger

This example demonstrates the configuration and use of One-shot pulse output mode with TAU_PWM timer when using the software trigger.

```
void tau_pwm_one_shot_software_trigger_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_one_shot_software_cfg);
```

```
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Enable triggering (including by software). */
   (void) R_TAU_PWM_Enable(&g_timer0_ctrl);
   g_timer_master_cycle_end_counter = 0;
   g_timer_slave_cycle_end_counter = 0;
   (void) R_TAU_PWM_Start(&g_timer0_ctrl);
   uint32_t timeout = TIMER_TEST_WAVEFORM_TIMEOUT;
while (((timeout--) > 0) && (0 == g_timer_slave_cycle_end_counter))
   {
/* Wait for one shot pulse output to complete */
   }
/* Ensure exactly one slave interrupt and one master interrupt have occurred. */
   assert(1 == g_timer_master_cycle_end_counter);
   assert(1 == g_timer_slave_cycle_end_counter);
   g_timer_master_cycle_end_counter = 0;
   g_timer_slave_cycle_end_counter = 0;
/* Start trigger by software. */
   (void) R_TAU_PWM_Start(&g_timer0_ctrl);
   timeout = TIMER_TEST_WAVEFORM_TIMEOUT;
while (((timeout--) > 0) && (0 == g_timer_slave_cycle_end_counter))
   {
/* Wait for one shot pulse output to complete */
   }
/* Ensure exactly one slave interrupt and one master interrupt have occurred. */
   assert(1 == g_timer_master_cycle_end_counter);
   assert(1 == g_timer_slave_cycle_end_counter);
/* Stop timer and disable the software trigger. */
   (void) R_TAU_PWM_Stop(&g_timer0_ctrl);
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_TAU_PWM_StatusGet(&g_timer0_ctrl, &status);
}
```

### TAU_PWM One-Shot Pulse Output Mode Example - Pin Input Trigger

This example demonstrates the configuration and use of One-shot pulse output mode with TAU_PWM timer when using an external pin input trigger.

```
void tau_pwm_one_shot_pin_input_trigger_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_one_shot_input_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the input trigger. */
    (void) R_TAU_PWM_Enable(&g_timer0_ctrl);
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
 /* Wait for trigger source from the input pin */
 /* Disable the input trigger. */
    (void) R_TAU_PWM_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_PWM_StatusGet(&g_timer0_ctrl, &status);
}
```

### TAU_PWM PWM Output Mode Example

This example demonstrates the configuration and use of PWM output mode with TAU_PWM timer.

```
void tau_pwm_multiple_pwm_output_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_pwm_output_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
```

```
    (void) R_TAU_PWM_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_PWM_StatusGet(&g_timer0_ctrl, &status);
}
```

### TAU_PWM Period Update Example

This an example of updating the period.

```
#define TAU_PWM_EXAMPLE_MSEC_PER_SEC (1000)
#define TAU_PWM_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void tau_pwm_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz) */
 timer_info_t info;
    (void) R_TAU_PWM_InfoGet(&g_timer0_ctrl, &info);
    uint32_t timer_freq_hz = info.clock_frequency;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX. A cast to uint32_t is used to prevent
this. */
    uint32_t period_counts =
       (uint32_t) ((timer_freq_hz * TAU_PWM_EXAMPLE_DESIRED_PERIOD_MSEC) /
TAU_PWM_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
```

```
  * period is larger than UINT16_MAX. */
    err = R_TAU_PWM_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

### TAU_PWM Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define TAU_PWM_EXAMPLE_MSEC_PER_SEC (1000)
#define TAU_PWM_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void tau_pwm_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz) */
 timer_info_t info;
    (void) R_TAU_PWM_InfoGet(&g_timer0_ctrl, &info);
    uint32_t timer_freq_hz = info.clock_frequency;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX. A cast to uint32_t is used to prevent
this. */
    uint32_t period_counts =
       (uint32_t) ((timer_freq_hz * TAU_PWM_EXAMPLE_DESIRED_PERIOD_MSEC) /
TAU_PWM_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
```

```
    err = R_TAU_PWM_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

### ELC Example

This is an example of using TAU_PWM with ELC events.

```
/* This example shows how to use ELC event to trigger the timer counters. */
void tau_pwm_elc_event_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_elc_event_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the elc event trigger. */
    (void) R_TAU_PWM_Enable(&g_timer0_ctrl);
    (void) R_TAU_PWM_Start(&g_timer0_ctrl);
 /* Wait for ELC event */
 /* Disable the elc event trigger. */
    (void) R_TAU_PWM_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_TAU_PWM_StatusGet(&g_timer0_ctrl, &status);
}
```

### Data Structures

| | |
|---|---|
| struct | tau_pwm_channel_cfg_t |
| struct | tau_pwm_extended_cfg_t |
| struct | tau_pwm_instance_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | tau_pwm_operation_clock_t |
| enum | tau_pwm_source_t |

| | |
|---|---|
| enum | tau_pwm_detect_edge_t |
| enum | tau_pwm_output_level_t |
| enum | tau_pwm_output_polarity_t |
| enum | tau_pwm_io_pin_t |

## Data Structure Documentation

### ◆ tau_pwm_channel_cfg_t

| struct tau_pwm_channel_cfg_t | | |
|---|---|---|
| TAU_PWM per channel configuration. | | |
| Data Fields | | |
| uint8_t | channel | Slave Channel Number (1..7) |
| uint16_t | duty_cycle_counts | One-shot: Pulse_width_counts; PWM: Duty_cycle_counts. |
| tau_pwm_output_level_t | output_level | Setting of output level for TAU. |
| tau_pwm_output_polarity_t | output_polarity | Setting of output polarity for TAU. |
| uint8_t | cycle_end_ipl | TAU slave channel IPL. |
| IRQn_Type | cycle_end_irq | TAU slave channel IRQ. |

### ◆ tau_pwm_extended_cfg_t

| struct tau_pwm_extended_cfg_t | | |
|---|---|---|
| Extended configuration structure for TAU_PWM | | |
| Data Fields | | |
| tau_pwm_operation_clock_t | operation_clock | Setting of operation clock for master and slave channels. |
| tau_pwm_source_t | trigger_source | Trigger source for master channel. |
| tau_pwm_detect_edge_t | detect_edge | Trigger edge to start pulse period measurement. |
| tau_pwm_channel_cfg_t const * | p_slave_channel_cfgs[TAU_PWM_MAX_NUM_SLAVE_CHANNELS] | Configuration for each slave channel, at least 1 slave channel is required. |

### ◆ tau_pwm_instance_ctrl_t

| struct tau_pwm_instance_ctrl_t |
|---|
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

## Enumeration Type Documentation

### ◆ tau_pwm_operation_clock_t

| enum tau_pwm_operation_clock_t | |
|---|---|
| Operation clock. | |
| Enumerator | |
| TAU_PWM_OPERATION_CLOCK_CK00 | Operation Clock CK00. |
| TAU_PWM_OPERATION_CLOCK_CK01 | Operation CLock CK01. |

### ◆ tau_pwm_source_t

| enum tau_pwm_source_t | |
|---|---|
| Trigger Source | |
| Enumerator | |
| TAU_PWM_SOURCE_PIN_INPUT | Use TI0n pin input as trigger source. |
| TAU_PWM_SOURCE_ELC_EVENT | Use ELC events as trigger source. |

### ◆ tau_pwm_detect_edge_t

| enum tau_pwm_detect_edge_t | |
|---|---|
| TI0n pin input edge | |
| Enumerator | |
| TAU_PWM_DETECT_EDGE_FALLING | Detects falling edge. |
| TAU_PWM_DETECT_EDGE_RISING | Detects rising edge. |
| TAU_PWM_DETECT_EDGES_BOTH | Detects both edges. |

### ◆ tau_pwm_output_level_t

| enum tau_pwm_output_level_t | |
|---|---|
| Level of TAU pin | |
| Enumerator | |
| TAU_PWM_OUTPUT_LEVEL_LOW | Pin level low. |
| TAU_PWM_OUTPUT_LEVEL_HIGH | Pin level high. |

### ◆ tau_pwm_output_polarity_t

| enum tau_pwm_output_polarity_t | |
|---|---|
| Timer output polarity | |
| Enumerator | |
| TAU_PWM_OUTPUT_POLARITY_ACTIVE_HIGH | Positive logic output (active-high) |
| TAU_PWM_OUTPUT_POLARITY_ACTIVE_LOW | Negative logic output (active-low) |

### ◆ tau_pwm_io_pin_t

| enum tau_pwm_io_pin_t | |
|---|---|
| Input/Output pins, used to select which duty cycle to update in R_TAU_PWM_DutyCycleSet(). | |
| Enumerator | |
| TAU_PWM_IO_PIN_CHANNEL_0 | I/O pin of channel 0. |
| TAU_PWM_IO_PIN_CHANNEL_1 | I/O pin of channel 1. |
| TAU_PWM_IO_PIN_CHANNEL_2 | I/O pin of channel 2. |
| TAU_PWM_IO_PIN_CHANNEL_3 | I/O pin of channel 3. |
| TAU_PWM_IO_PIN_CHANNEL_4 | I/O pin of channel 4. |
| TAU_PWM_IO_PIN_CHANNEL_5 | I/O pin of channel 5. |
| TAU_PWM_IO_PIN_CHANNEL_6 | I/O pin of channel 6. |
| TAU_PWM_IO_PIN_CHANNEL_7 | I/O pin of channel 7. |

## Function Documentation

### ◆ R_TAU_PWM_Open()

fsp_err_t R_TAU_PWM_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the timer module and applies configurations. Implements timer_api_t::open.

The TAU_PWM implementation of the timer requires a tau_pwm_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
   err = R_TAU_PWM_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful |
| FSP_ERR_ASSERTION | A required input pointer is NULL, or one of the following is invalid: the count of source divider/period/duty cycle/slave channels, or the trigger source. |
| FSP_ERR_ALREADY_OPEN | Module is already open. |
| FSP_ERR_IRQ_BSP_DISABLED | ISR of master channel must be enabled |
| FSP_ERR_INVALID_MODE | Invalid configuration option provided for selected timer mode |
| FSP_ERR_INVALID_CHANNEL | The master/slave channel selected is not available on this device, slave channel number must be greater than master channel number. |

### ◆ R_TAU_PWM_Stop()

fsp_err_t R_TAU_PWM_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_TAU_PWM_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PWM_Start()

fsp_err_t R_TAU_PWM_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer (pwm mode) or triggers the one-shot pulse output by software (one-shot mode). Implements timer_api_t::start.

***Note***

*In one-shot mode, this function is supported only after the timer has been placed into the start trigger detection wait state by calling timer_api_t::enable*

Example:

```
/* Start the timer. */
   (void) R_TAU_PWM_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer successfully started. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_NOT_ENABLED | In One-shot mode, timer must be enabled first. |

### ◆ R_TAU_PWM_Reset()

fsp_err_t R_TAU_PWM_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the current period and duty cycle. Implements timer_api_t::reset.

***Note***

*If the timer is stopped when calling this function, the timer counter is not reset. The counter will be reset one cycle after the timer is next started (or restarted), since it takes one cycle to reload the initial count when starting the timer.*

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PWM_Enable()

fsp_err_t R_TAU_PWM_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event inputs that can start the counter and enables the software trigger. After a successful call to this function, the timer is placed into the trigger detection wait state. Implements timer_api_t::enable.

Example:

```
/* Enable triggering (including by software). */
   (void) R_TAU_PWM_Enable(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_MODE | The mode is invalid, only called in TIMER_MODE_ONE_SHOT mode. |
| FSP_ERR_UNSUPPORTED | Unsupported when one shot mode support is disabled |

### ◆ R_TAU_PWM_PeriodSet()

fsp_err_t R_TAU_PWM_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Sets period value provided. If the timer is running, the period will be updated after the next counter underflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

Example:

```
/* Get the source clock frequency (in Hz) */
timer_info_t info;
   (void) R_TAU_PWM_InfoGet(&g_timer0_ctrl, &info);
   uint32_t timer_freq_hz = info.clock_frequency;
/* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX. A cast to uint32_t is used to prevent
this. */
   uint32_t period_counts =
      (uint32_t) ((timer_freq_hz * TAU_PWM_EXAMPLE_DESIRED_PERIOD_MSEC) /
TAU_PWM_EXAMPLE_MSEC_PER_SEC);
/* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
   err = R_TAU_PWM_PeriodSet(&g_timer0_ctrl, period_counts);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value written successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Period counts is out of range. |

### ◆ R_TAU_PWM_CompareMatchSet()

fsp_err_t R_TAU_PWM_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | TAU PWM compare match is not supported. |

### ◆ R_TAU_PWM_DutyCycleSet()

fsp_err_t R_TAU_PWM_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Duty cycle is updated in the timer data register. The updated duty cycle is reflected after the next cycle end (counter underflow).

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_TAU_PWM_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. */
   uint16_t duty_cycle_counts = (uint16_t) ((current_period_counts *
TAU_PWM_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                          TAU_PWM_EXAMPLE_MAX_PERCENT);
/* Set the calculated duty cycle. */
   err = R_TAU_PWM_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, 1);
   assert(FSP_SUCCESS == err);
```

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block. |
| [in] | duty_cycle_counts | Duty cycle to set in counts. |
| [in] | pin | Use tau_pwm_io_pin_t to select the target slave channel |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl was NULL or the pin is not one of tau_pwm_io_pin_t. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle is out of range or larger than period. |

### ◆ R_TAU_PWM_InfoGet()

fsp_err_t R_TAU_PWM_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_TAU_PWM_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, frequency written to caller's structure successfully |
| FSP_ERR_ASSERTION | p_ctrl or p_info was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PWM_StatusGet()

fsp_err_t R_TAU_PWM_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Retrieves the current timer state and master channel counter value and stores them in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_TAU_PWM_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current timer state and counter value retrieved successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_status was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PWM_CallbackSet()

fsp_err_t R_TAU_PWM_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | p_ctrl or p_callback was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_TAU_PWM_Close()

fsp_err_t R_TAU_PWM_Close ( timer_ctrl_t *const *p_ctrl*)

Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful close. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

## 5.2.19.10 Timer, Ultra Low-Power (r_ulpt)

Modules » Timers

**Functions**

| | |
|---|---|
| fsp_err_t | R_ULPT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| fsp_err_t | R_ULPT_Start (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ULPT_Stop (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ULPT_Reset (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ULPT_Enable (timer_ctrl_t *const p_ctrl) |
| fsp_err_t | R_ULPT_Disable (timer_ctrl_t *const p_ctrl) |

| | |
|---|---|
| fsp_err_t | R_ULPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts) |
| fsp_err_t | R_ULPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| fsp_err_t | R_ULPT_CompareMatchSet (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| fsp_err_t | R_ULPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| fsp_err_t | R_ULPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| fsp_err_t | R_ULPT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_ULPT_Close (timer_ctrl_t *const p_ctrl) |

### Detailed Description

Driver for the ULPT peripheral on RA MCUs. This module implements the Timer Interface.

# Overview

### Features

The ULPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, LOCO, SUBCLK, and external sources input to ULPTEVIn.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports counting based on an external clock input to ULPTEVIn.
- Supports debounce filter on ULPTEVIn pins.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.

### Selecting a Timer

RA MCUs can have multiple timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer, and the Ultra Low Power Timer (ULPT). When selecting between them, consider these factors:

| | GPT | AGT | ULPT |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Low Power Modes | The GPT can operate in sleep mode. | The AGT can operate in all low power modes (when count source is LOCO or subclock). | The ULPT can operate in all low power up to DSTBYs (when count source is LOCO or subclock). |
| Available Channels | The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels. | All MCUs have 2 AGT channels. | The number of ULPT channels is device specific, 2 is common. |
| Timer Resolution | All MCUs have at least one 32-bit GPT timer. | The AGT timers are 16-bit timers. | The ULPT timers are 32-bit timers. |
| Clock Source | The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses. | The AGT runs off PCLKB, LOCO, or subclock with a configurable divider up to 8 for PCLKB or up to 128 for LOCO or subclock. | The ULPT runs off PCLKB. LOCO, or subclock are count sources with a configurable divider up to 128. |

# Configuration

### Build Time Configurations for r_ulpt

The following build time configurations are defined in fsp_cfg/r_ulpt_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Pin Output Support | • Disabled<br>• Enabled | Disabled | If selected code for outputting a waveform to a pin is included in the build. |
| Pin Input Support | • Disabled<br>• Enabled (Required when Count Source is EVI on any channel) | Disabled | Enable input support to use ULPTEVin as count source |

### Configurations for Timers > Timer, Ultra-Low-Power (r_ulpt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, Ultra-Low-Power (r_ulpt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

| | | | |
|---|---|---|---|

| Configuration | Options | Default | Description |
|---|---|---|---|
| General | | | |
| Name | Name must be a valid C symbol | g_timer0 | Module name. |
| Channel | Channel number does not exist | 0 | Physical hardware channel. |
| Mode | • Periodic<br>• One-Shot<br>• PWM | Periodic | Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused. |
| Period | Value must be non-negative | 0x10000 | Specify the timer period based on the selected unit.<br><br>When the unit is set to 'Raw Counts', setting the period to 0x10000 results in the maximum period at the lowest divisor (fastest timer tick). Set the period to 0x10000 for a free running timer, pulse width measurement or pulse period measurement. Setting the period higher will automatically select a higher divider; the period can be set up to 0x80000 when counting from PCLKB or 0x800000 when counting from LOCO/subclock, which will use a divider of 8 or 128 respectively with the maximum period.<br><br>If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a |

| | | | |
|---|---|---|---|
| | | | comment in the timer_cfg_t structure. |
| Period Unit | • Raw Counts<br>• Nanoseconds<br>• Microseconds<br>• Milliseconds<br>• Seconds<br>• Hertz<br>• Kilohertz | Raw Counts | Unit of the period specified above |
| Count Source | • LOCO<br>• SUBCLOCK<br>• EVI (Requires Pin Input Support-Enabled) | LOCO | ULPT count source. NOTE: The divisor is calculated automatically based on the selected period. See ulpt_count_source_t documentation for details. |
| Output | | | |
| Duty Cycle Percent (only applicable in PWM mode) | Value must be between 0 and 100 | 50 | Specify the timer duty cycle percent. Only used in PWM mode. |
| ULPTOA Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure Match ULPTOA output. |
| ULPTOB Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure Match ULPTOB output. |
| ULPTO Output | • Disabled<br>• Start Level Low<br>• Start Level High | Disabled | Configure Pulse ULPTO output. |
| Input | | | |
| Input Filter | • No Filter<br>• Filter sampled at PCLKB<br>• Filter sampled at PCLKB / 8<br>• Filter sampled at PCLKB / 32 | No Filter | ULPTEVIn filter. Only applies if the count source is ULPTEVIn, event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. |
| Enable Pin | • Enable Function Ignored<br>• Enable Function Low<br>• Enable Function High<br>• Enable Function | Enable Function Ignored | ULPTEEn enable edge. Only applies if the count source is ULPTEVIn, event counter mode |

| | | | |
|---|---|---|---|
| | Start<br>• Enable Function Restart | | |
| Trigger Edge | • Trigger Edge Rising<br>• Trigger Edge Falling<br>• Trigger Edge Both | Trigger Edge Rising | ULPTEVIn trigger edge. Applies Only applies if the count source is ULPTEVIn, event counter mode. |
| Event Edge | • Event Edge Rising<br>• Event Edge Falling<br>• Event Edge Both | Event Edge Rising | Select the ULPTEVin edge. Applies if count input is ULPTEVin. |
| Interrupts | | | |
| Callback | Name must be a valid C symbol | NULL | A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses. |
| Underflow Interrupt Priority | MCU Specific Options | | Timer interrupt priority. |

### Clock Configuration

The ULPT subsystem is driven by the PCLKB, but countdown is driven LOCO, Subclock, or event input. You can set the clock frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

### Pin Configuration

This module can use the ULPTO, ULPTOA and ULPTOB pins as output pins for periodic, one-shot, or PWM signals.

For event counting, the input clocking signal must be applied to the ULPTEVIn pin.

### Timer Period

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units, and clock speed.

When the selected unit is "Raw counts", the maximum allowed period setting varies depending on the selected clock source:

| Clock source | Maximum period (counts) |
|---|---|
| LOCO/Subclock | 0xFFFFFFFF |

| | |
|---|---|
| All other sources | 0xFFFFFFFF |

*Note*

*Though the ULPT is a 32-bit timer, because the period interrupt occurs when the counter underflows, setting the period register to 0 results in an effective period of 1 count. For this reason all user-provided raw count values reflect the actual number of period counts (not the raw register values).*

# Usage Notes

### Starting and Stopping the ULPT

After starting or stopping the timer, ULPT registers cannot be accessed until the ULPT state is updated after 3 ULPTCLK cycles. If another ULPT function is called before the 3 ULPTCLK period elapses, the function spins waiting for the ULPT state to update. The required wait time after starting or stopping the timer can be determined using the frequency of ULPTCLK, which is derived from timer_cfg_t::source_div and ulpt_extended_cfg_t::count_source.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running ULPT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

### Low Power Modes

The ULPT can be used to enter snooze mode or to wake the MCU from snooze, software standby, or deep software standby modes when a counter underflow occurs. The compare match A and B events can also be used to wake from software standby or snooze modes.

### One-Shot Mode

The ULPT timer does support one-shot mode natively. The interrupt with put the module in the stop state. The callback is only called once in this case. If needed the timer needs to be re-started via the driver call.

### Periodic Output

The ULPTO toggles each time the counter underflows.

Examples of periodic signals that can be generated by this module are shown below:

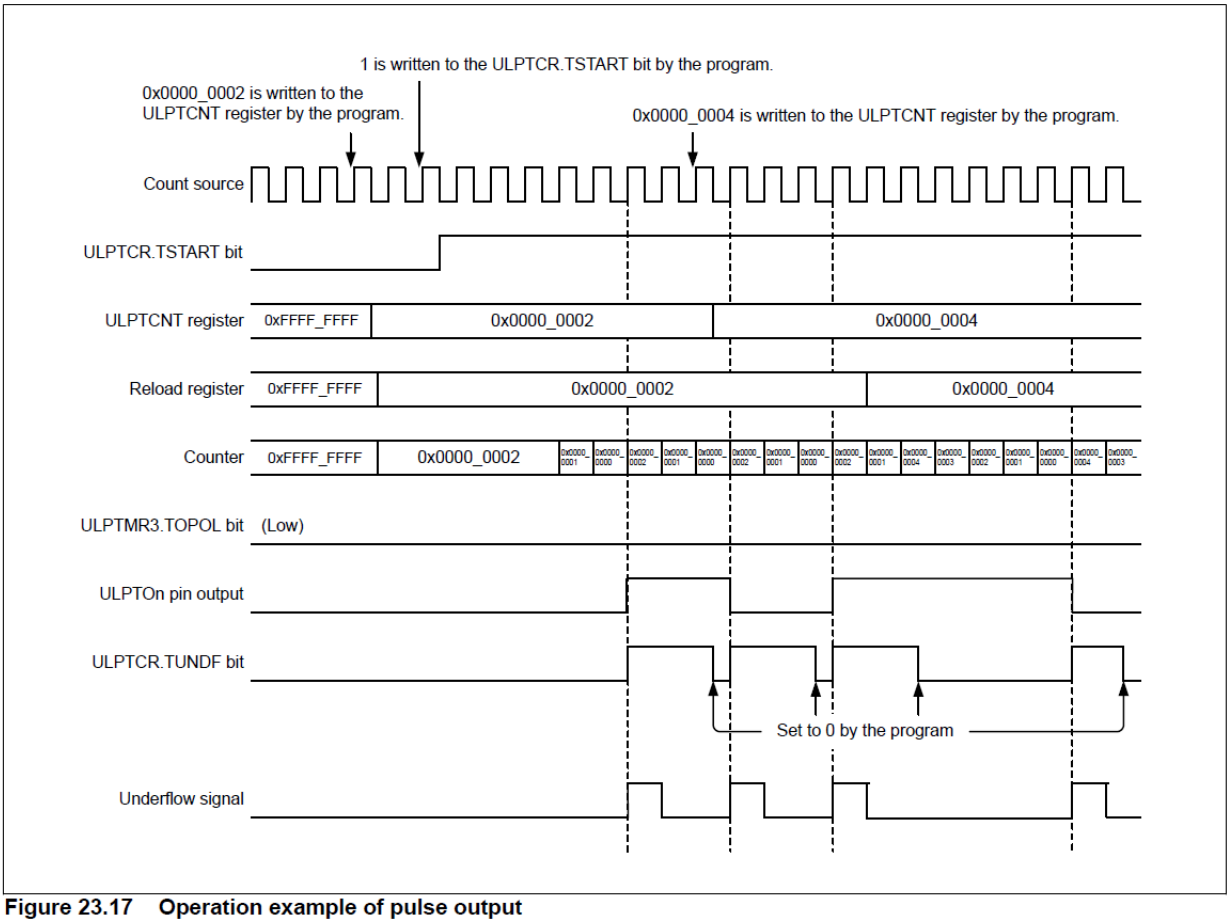



**Figure 23.17 Operation example of pulse output**

Figure 346: ULPT Periodic Output

### PWM Output

The ULPTOA or ULPTOB pin toggles each time the compare match timer matches the down counter. It also toggles once the underflow occurs in periodic mode. for example setting the ULPT counter to 0x1000, and compare match A to half (0x800) will created a 50% duty cycle output on ULPTOA. Setting ULPTB to a quarter of that (0x400) will create a 25% duty cycle wave. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

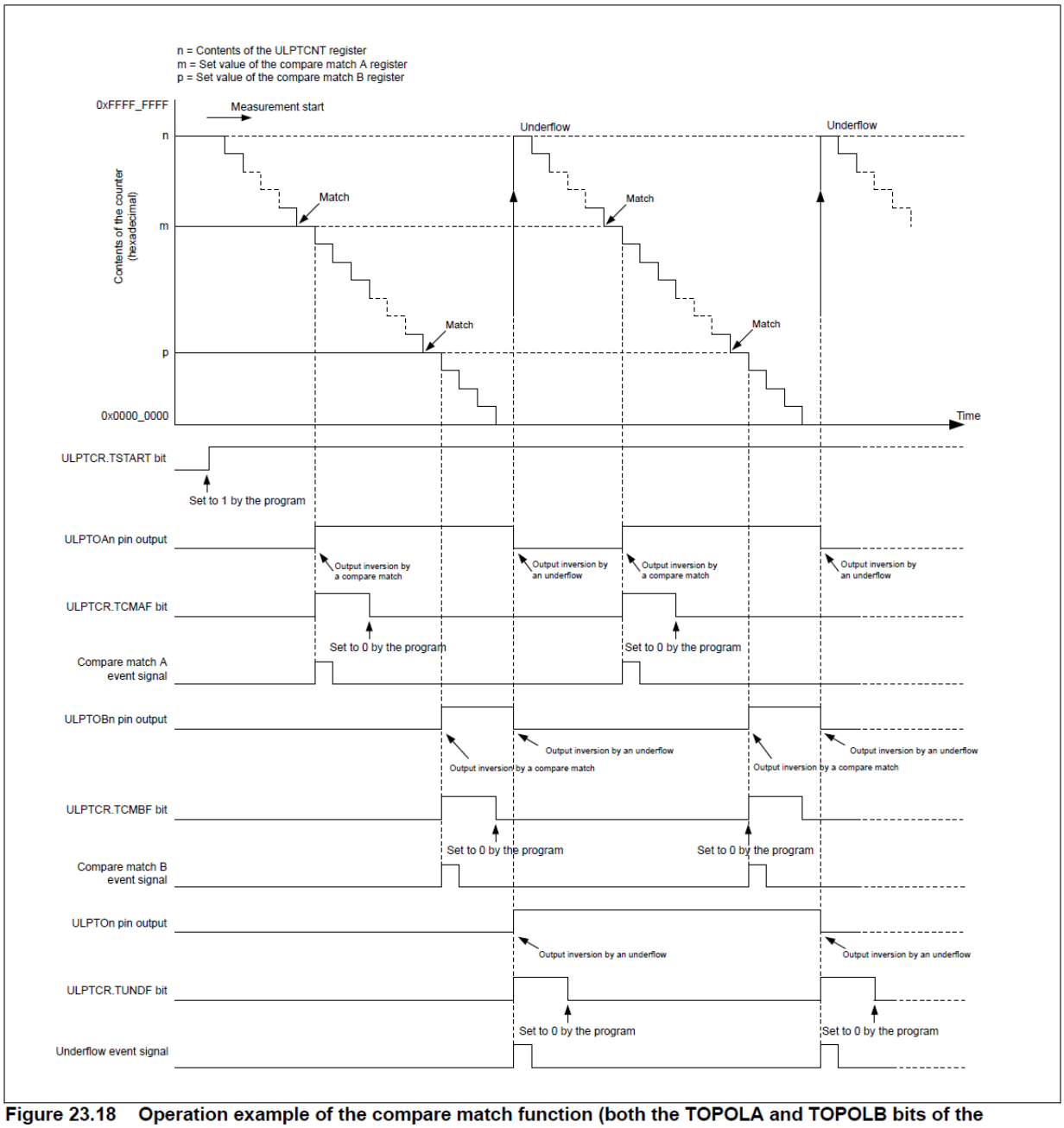



**Figure 23.18 Operation example of the compare match function (both the TOPOLA and TOPOLB bits of the ULPTCMSR register are set to 0)**

Figure 347: ULPT Periodic Output

### Triggering ELC Events with ULPT

The ULPT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

# Examples

### ULPT Basic Example

This is a basic example of minimal use of the ULPT in an application.

```
void ulpt_basic_example (void)
{
 fsp_err_t      err = FSP_SUCCESS;
 timer_status_t status;
 /* Initializes the module. */
    err = R_ULPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_ULPT_Start(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
    (void) R_ULPT_StatusGet(&g_timer0_ctrl, &status);
}
```

### ULPT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

### ULPT Free Running Counter Example

To use the ULPT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF.

```
void ulpt_counter_example (void)
{
 fsp_err_t      err = FSP_SUCCESS;
```

```
 timer_status_t status;
 /* Initializes the module. */
    err = R_ULPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_ULPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. This will set the counter back to max */
    (void) R_ULPT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
    (void) R_ULPT_StatusGet(&g_timer0_ctrl, &status);
}
```

### ULPT Period Update Example

This an example of updating the period.

```
#define ULPT_EXAMPLE_MSEC_PER_SEC (1000)
#define ULPT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void ulpt_period_calculation_example (void)
{
 fsp_err_t    err = FSP_SUCCESS;
 timer_info_t ulptInfo;
    uint32_t    timer_freq_hz = 0;
 /* Initializes the module. */
    err = R_ULPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_ULPT_Start(&g_timer0_ctrl);
    err = R_ULPT_Enable(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Get the source clock frequency (in Hz).
  * - Use the R_ULPT_InfoGet function (it accounts for the clock source and divider).
```

```
*/
 if (R_ULPT_InfoGet(&g_timer0_ctrl, &ulptInfo) == FSP_SUCCESS)
    {
       timer_freq_hz = ulptInfo.clock_frequency;
    }
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * ULPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
ULPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_ULPT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
    err = R_ULPT_Disable(&g_timer0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### ULPT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define ULPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define ULPT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void ulpt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ULPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Start the timer. */
    (void) R_ULPT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_ULPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. */
    uint32_t duty_cycle_counts = (current_period_counts *
ULPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                 ULPT_EXAMPLE_MAX_PERCENT;
 /* Set the calculated duty cycle. */
    err = R_ULPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts,
ULPT_OUTPUT_PIN_ULPTOA);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | ulpt_instance_ctrl_t |
| struct | ulpt_extended_cfg_t |

## Enumerations

| | |
|---|---|
| enum | ulpt_clock_t |
| enum | ulpt_enable_function_t |
| enum | ulpt_trigger_edge_t |
| enum | ulpt_event_pin_t |
| enum | ulpt_output_pin_t |
| enum | ulpt_pulse_pin_cfg_t |
| enum | ulpt_match_pin_cfg_t |
| enum | ulpt_ulptevi_filter_t |

## Data Structure Documentation

### ◆ ulpt_instance_ctrl_t

| struct ulpt_instance_ctrl_t |
| --- |
| Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called. |

### ◆ ulpt_extended_cfg_t

| struct ulpt_extended_cfg_t | | |
| --- | --- | --- |
| Optional ULPT extension data structure. | | |
| Data Fields | | |
| ulpt_clock_t | count_source | ULPT channel clock source. |
| ulpt_ulptevi_filter_t | ulptevi_filter | Input filter for ULTPEVI. |
| ulpt_enable_function_t | enable_function | Counter function when ULPTEE is valid. |
| ulpt_trigger_edge_t | trigger_edge | Enable trigger edge (start and restart functions only). |
| ulpt_event_pin_t | event_pin | Event pin (event counting only). |
| ulpt_pulse_pin_cfg_t | ulpto | Pulse output pin. |
| union ulpt_extended_cfg_t | __unnamed__ | |

## Enumeration Type Documentation

### ◆ ulpt_clock_t

| enum ulpt_clock_t | |
| --- | --- |
| Count source. | |
| Enumerator | |
| ULPT_CLOCK_LOCO | LOCO count source, division by 1, 2, 4, 8, 16, 32, 64, 128. |
| ULPT_CLOCK_SUBCLOCK | Subclock count source, division by 1, 2, 4, 8, 16, 32, 64, 128. |
| ULPT_CLOCK_ULPTEVI | Counts external events on ULPTEVI. |

### ◆ ulpt_enable_function_t

| enum ulpt_enable_function_t | |
|---|---|
| Counter mode for event enable. | |
| Enumerator | |
| ULPT_ENABLE_FUNCTION_IGNORED | Always count external events, ignore ULPTEE. |
| ULPT_ENABLE_FUNCTION_ENABLE_LOW | Event counting is enabled while ULPTEE is low (event counting only). |
| ULPT_ENABLE_FUNCTION_ENABLE_HIGH | Event counting is enabled while ULPTEE is high (event counting only). |
| ULPT_ENABLE_FUNCTION_START | Counting is started after ULPTEE. |
| ULPT_ENABLE_FUNCTION_RESTART | Counting is restarted after ULPTEE. |

### ◆ ulpt_trigger_edge_t

| enum ulpt_trigger_edge_t | |
|---|---|
| Enable signal trigger edge for start and restart functions. | |
| Enumerator | |
| ULPT_TRIGGER_EDGE_RISING | Timer enable function occurs on the rising edge of ULPTEE. |
| ULPT_TRIGGER_EDGE_FALLING | Timer enable function occurs on the falling edge of ULPTEE. |
| ULPT_TRIGGER_EDGE_BOTH | Timer enable function occurs on any edge of ULPTEE. |

### ◆ ulpt_event_pin_t

| enum ulpt_event_pin_t | |
|---|---|
| Event signal pin. | |
| Enumerator | |
| ULPT_EVENT_PIN_RISING | Event count occurs on the rising edge. |
| ULPT_EVENT_PIN_FALLING | Event count occurs on the falling edge. |
| ULPT_EVENT_PIN_BOTH | Event count occurs on both edges. |

### ◆ ulpt_output_pin_t

| enum ulpt_output_pin_t | |
|---|---|
| Output pins, used to select which duty cycle to update in R_ULPT_DutyCycleSet(). | |
| Enumerator | |
| ULPT_OUTPUT_PIN_ULPTOA | Compare match A output. |
| ULPT_OUTPUT_PIN_ULPTOB | Compare match B output. |

### ◆ ulpt_pulse_pin_cfg_t

| enum ulpt_pulse_pin_cfg_t | |
|---|---|
| ULPTO pulse output pin. | |
| Enumerator | |
| ULPT_PULSE_PIN_CFG_DISABLED | Output pin disabled. |
| ULPT_PULSE_PIN_CFG_ENABLED_START_LEVEL_LOW | Output pin Enabled Start Low. |
| ULPT_PULSE_PIN_CFG_ENABLED_START_LEVEL_HIGH | Output pin enabled Start Hig. |

### ◆ ulpt_match_pin_cfg_t

| enum ulpt_match_pin_cfg_t | |
|---|---|
| ULPT match output pin. | |
| Enumerator | |
| ULPT_MATCH_PIN_CFG_DISABLED | Match output disabled. |
| ULPT_MATCH_PIN_CFG_START_LEVEL_LOW | Match output enabled, starts low. |
| ULPT_MATCH_PIN_CFG_START_LEVEL_HIGH | Match output enabled, starts high. |

### ◆ ulpt_ulptevi_filter_t

| enum ulpt_ulptevi_filter_t | |
|---|---|
| Input filter, applied to ULPTEVI in event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency. | |
| Enumerator | |
| ULPT_ULPTEVI_FILTER_NONE | No filter. |
| ULPT_ULPTEVI_FILTER_PCLKB | Filter at PCLKB. |
| ULPT_ULPTEVI_FILTER_PCLKB_DIV_8 | Filter at PCLKB / 8. |
| ULPT_ULPTEVI_FILTER_PCLKB_DIV_32 | Filter at PCLKB / 32. |

## Function Documentation

### ◆ R_ULPT_Open()

fsp_err_t R_ULPT_Open ( timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg* )

Initializes the ULPT module instance. Implements timer_api_t::open.

The ULPT implementation of the general timer can accept an optional ulpt_extended_cfg_t extension parameter. For ULPT, the extension specifies the clock to be used as timer source and the output pin configurations. If the extension parameter is not specified (NULL), the default clock LOCO is used and the output pins are disabled.

Example:

```
/* Initializes the module. */
   err = R_ULPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Initialization was successful and timer has started. |
| FSP_ERR_ASSERTION | A required input pointer is NULL or the period is not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_ALREADY_OPEN | R_ULPT_Open has already been called for this p_ctrl. |
| FSP_ERR_IRQ_BSP_DISABLED | A required interrupt has not been enabled in the vector table. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | Requested channel number is not available on ULPT. |

### ◆ R_ULPT_Start()

fsp_err_t R_ULPT_Start ( timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_ULPT_Start(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer started. |
| FSP_ERR_ASSERTION | p_ctrl is null. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_Stop()

fsp_err_t R_ULPT_Stop ( timer_ctrl_t *const *p_ctrl*)

Stops the timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. This will set the counter back to max */
   (void) R_ULPT_Stop(&g_timer0_ctrl);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer stopped. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_Reset()

fsp_err_t R_ULPT_Reset ( timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the period minus one. Implements timer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter reset. |
| FSP_ERR_ASSERTION | p_ctrl is NULL |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_Enable()

fsp_err_t R_ULPT_Enable ( timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
err = R_ULPT_Enable(&g_timer0_ctrl);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully enabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_ULPT_Disable()

fsp_err_t R_ULPT_Disable ( timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Example:

```
err = R_ULPT_Disable(&g_timer0_ctrl);
assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | External events successfully disabled. |
| FSP_ERR_ASSERTION | p_ctrl was NULL. |
| FSP_ERR_NOT_OPEN | The instance is not opened. |

### ◆ R_ULPT_PeriodSet()

fsp_err_t R_ULPT_PeriodSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts* )

Updates period. The new period is updated immediately and the counter is reset to the maximum value. Implements timer_api_t::periodSet.

**Warning**

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and an AGT underflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter underflow after processing completes.
Stop the timer before calling this function if one-shot output is used.

Example:

```
/* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
 * period is larger than UINT16_MAX. */
  err = R_ULPT_PeriodSet(&g_timer0_ctrl, period_counts);
  assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period value updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the period was not in the valid range of 1 to 0xFFFF. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_DutyCycleSet()

fsp_err_t R_ULPT_DutyCycleSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin* )

Updates duty cycle. If the timer is counting, the new duty cycle is reflected after the next counter underflow. Implements timer_api_t::dutyCycleSet.

Example:

```
/* Set the calculated duty cycle. */
   err = R_ULPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts,
ULPT_OUTPUT_PIN_ULPTOA);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Duty cycle updated. |
| FSP_ERR_ASSERTION | A required pointer was NULL, or the pin was not ULPT_ULPTO_ULPTOA or ULPT_ULPTO_ULPTOB. |
| FSP_ERR_INVALID_ARGUMENT | Duty cycle was not in the valid range of 0 to period (counts) - 1 |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |
| FSP_ERR_UNSUPPORTED | ULPT_CFG_OUTPUT_SUPPORT_ENABLE is 0. |

### ◆ R_ULPT_CompareMatchSet()

fsp_err_t R_ULPT_CompareMatchSet ( timer_ctrl_t *const *p_ctrl*, uint32_t const *compare_match_value*, timer_compare_match_t const *match_channel* )

Placeholder for unsupported compareMatch function. Implements timer_api_t::compareMatchSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | ULPT compare match is not supported. |

### ◆ R_ULPT_InfoGet()

fsp_err_t R_ULPT_InfoGet ( timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info* )

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* Get the source clock frequency (in Hz).
 * - Use the R_ULPT_InfoGet function (it accounts for the clock source and divider).
*/
if (R_ULPT_InfoGet(&g_timer0_ctrl, &ulptInfo) == FSP_SUCCESS)
   {
      timer_freq_hz = ulptInfo.clock_frequency;
   }
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Period, count direction, and frequency stored in p_info. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_StatusGet()

fsp_err_t R_ULPT_StatusGet ( timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status* )

Retrieves the current state and counter value stores them in p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
   (void) R_ULPT_StatusGet(&g_timer0_ctrl, &status);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Current status and counter value provided in p_status. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

### ◆ R_ULPT_CallbackSet()

fsp_err_t R_ULPT_CallbackSet ( timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void *const *p_context*, timer_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |
| FSP_ERR_NO_CALLBACK_MEMORY | p_callback is non-secure and p_callback_memory is either secure or NULL. |

### ◆ R_ULPT_Close()

fsp_err_t R_ULPT_Close ( timer_ctrl_t *const *p_ctrl*)

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Timer closed. |
| FSP_ERR_ASSERTION | p_ctrl is NULL. |
| FSP_ERR_NOT_OPEN | The instance control structure is not opened. |

## 5.2.20 Transfer

Modules

### Detailed Description

Transfer Modules.

### Modules

| | |
|---|---|
| Transfer (r_dmac) | Driver for the DMAC peripheral on RA MCUs. This module implements |

| | |
|---|---|
| | the Transfer Interface. |
| | Transfer (r_dtc)<br>Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface. |

### 5.2.20.1 Transfer (r_dmac)

Modules » Transfer

#### Functions

| | |
|---|---|
| fsp_err_t | R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) |
| fsp_err_t | R_DMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info) |
| fsp_err_t | R_DMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers) |
| fsp_err_t | R_DMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) |
| fsp_err_t | R_DMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info) |
| fsp_err_t | R_DMAC_Reload (transfer_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers) |
| fsp_err_t | R_DMAC_CallbackSet (transfer_ctrl_t *const p_api_ctrl, void(*p_callback)(dmac_callback_args_t *), void *const p_context, dmac_callback_args_t *const p_callback_memory) |
| fsp_err_t | R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl) |

#### Detailed Description

Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface.

# Overview

The Direct Memory Access Controller (DMAC) transfers data from one memory location to another without using the CPU.

### Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
  - Repeat-Block transfer (Not available on all MCUs)
- Address increment, decrement, fixed, or offset modes
- Triggered by ELC events
  - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

# Configuration

### Build Time Configurations for r_dmac

The following build time configurations are defined in fsp_cfg/r_dmac_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |

### Configurations for Transfer > Transfer (r_dmac)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dmac).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid C symbol | g_transfer0 | Module name. |
| Channel | Value must be a non-negative integer | 0 | Specify the hardware channel. |
| Mode | MCU Specific Options | | Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of |

| | | | |
|---|---|---|---|
| | | | Blocks; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks. |
| Transfer Size | • 1 Byte<br>• 2 Bytes<br>• 4 Bytes | 2 Bytes | Select the transfer size. |
| Destination Address Mode | • Fixed<br>• Offset addition<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the destination. |
| Source Address Mode | • Fixed<br>• Offset addition<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the source. |
| Repeat Area (Unused in Normal Mode) | • Destination<br>• Source | Source | Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode. |
| Number of Transfers | Value must be a non-negative integer | 1 | Specify the number of transfers for repeat and normal mode or block size for repeat-block transfer mode. |
| Number of Blocks (Valid only in Repeat,Block or Repeat-Block Mode) | Value must be a non-negative integer | 0 | Specify the number of blocks to transfer in Repeat,Block or Repeat-Block mode. |
| Activation Source | MCU Specific Options | | Select the DMAC transfer start event. If no ELC event is chosen then software start can be used. |
| Callback | Name must be a valid C symbol | NULL | A user callback that is called at the end of the transfer. |
| Transfer End Interrupt Priority | MCU Specific Options | | Select the transfer end interrupt priority. |
| Interrupt Frequency | • Interrupt after all transfers have completed<br>• Interrupt after each block, or | Interrupt after all transfers have completed | Select to have interrupt after each transfer or after last transfer. |

| | | | |
|---|---|---|---|
| | repeat size is transferred | | |
| Offset value (Valid only when address mode is Offset addition) | Value must be a 24 bit signed integer. | 1 | Offset value is added to the address after each transfer. |
| Source Buffer Size | Value must be a non-negative integer with a maximum configurable value of 65535. | 1 | Specify the size of whole source buffer (valid only for Repeat-Block transfer mode with source address update mode other than offset addition). |

### Clock Configuration

The DMAC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build, or by using the CGC module at run-time.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Source and Destination Configuration

R_DMAC_Reset() API function should be called to set the source and destination before starting transfer operation.

### Transfer Modes

The DMAC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time the configured ELC event is received by the DMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment, decrement, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the ELC event and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area (source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the block count reaches 0, transfers will no longer be triggered by the ELC event and the CPU may be interrupted to signal that all transfers have finished.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.
- **Repeat-Block Mode** - In repeat-block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can be configured to a 16 bit number. If the destination address mode is offset mode,

maximum configurable number of blocks is 0xFFFF for block size(length) of one with data transfer size as byte,0x7FFF for block size of one with data transfer size as half word and 0x3FFF for block size of one with data size as word. After each block transfer the source address and the destination address will be incremented or decremented to the next block address. In case of offset address mode for source address, the source address size is the total size of source buffer after which the source area is rolled over, block size can be smaller than the source buffer size.For source address mode as offset mode, the maximum configurable source buffer size is 0xFFFF for transfer data size of a byte,0x7FFF for transfer data size of half word and 0x3FFF for transfer data size of word. Repeat-block mode can be used to implement single ring buffer to multiple ring buffer transfer type design.

**Selecting the DTC or DMAC**

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

| | DTC | DMAC |
|---|---|---|
| Repeat Mode | • Repeats forever<br>• Max repeat size is 256 x 4 bytes | • Configurable number of repeats<br>• Max repeat size is 1024 x 4 bytes |
| Block Mode | • Max block size is 256 x 4 bytes | • Max block size is 1024 x 4 bytes |
| Channels | • One instance per interrupt | • MCU specific (8 channels or less) |
| Chained Transfers | • Supported | • Not Supported |
| Software Trigger | • Must use the software ELC event | • Has support for software trigger without using software ELC event<br>• Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT |
| Offset Address Mode | • Not supported | • Supported |

**Interrupts**

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

## Normal Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | N/A |
| TRANSFER_IRQ_END | Interrupt after last transfer | Interrupt after last transfer |

## Repeat Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | Interrupt after each repeat |
| TRANSFER_IRQ_END | Interrupt after each repeat | Interrupt after last transfer |

### Block Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each block | Interrupt after each block |
| TRANSFER_IRQ_END | Interrupt after last block | Interrupt after last block |

### Repeat-block Mode

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | N/A | N/A |
| TRANSFER_IRQ_END | N/A | Interrupt after last block |

#### Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

#### Offset Address Mode

When the source or destination mode is configured to offset mode, a configurable offset is added to the source or destination pointer after each transfer. The offset is a signed 24 bit number.

## Examples

### Basic Example

This is a basic example of minimal use of the DMAC in an application. In this case, one or more events have been routed to the DMAC for handling so it only needs to be enabled to start accepting transfers.

```
void dmac_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the DMAC so that it responds to transfer requests. */
```

```
    err = R_DMAC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

### CRC32 Example

In this example the DMAC is used to feed the CRC peripheral to perform a CRC32 operation.

```
volatile bool g_transfer_complete = false;
void dmac_callback (dmac_callback_args_t * cb_data)
{
 FSP_PARAMETER_NOT_USED(cb_data);
    g_transfer_complete = true;
}
void dmac_crc_example (void)
{
    uint8_t p_src[TRANSFER_LENGTH];
 /* Initialize p_src to [ABC..OP] */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       p_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Set transfer source address to p_src */
    g_transfer_cfg.p_info->p_src = (void *) p_src;
 /* Set transfer destination address to the CRC data input register */
    g_transfer_cfg.p_info->p_dest = (void *) &R_CRC->CRCDIR;
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable DMAC transfers. */
    (void) R_DMAC_Enable(&g_transfer_ctrl);
 /* Open the CRC module. */
    err = R_CRC_Open(&g_crc_ctrl, &g_crc_cfg);
    assert(FSP_SUCCESS == err);
```

```
/* Clear the transfer complete flag. */
   g_transfer_complete = false;
/* Trigger the transfer using software. */
   err = R_DMAC_SoftwareStart(&g_transfer_ctrl, TRANSFER_START_MODE_SINGLE);
   assert(FSP_SUCCESS == err);
while (!g_transfer_complete)
   {
/* Wait for transfer complete interrupt */
   }
/* Get CRC result and perform final XOR. */
   uint32_t crc32;
   (void) R_CRC_CalculatedValueGet(&g_crc_ctrl, &crc32);
   crc32 ^= CRC32_FINAL_XOR_VALUE;
/* Verify that the CRC32 is calculated correctly. */
/* CRC32("ABCD...NOP") = 0xE0E8FF4D. */
const uint32_t expected_crc32 = 0xE0E8FF4D;
if (expected_crc32 != crc32)
   {
/* Handle any CRC errors. This function should be defined by the user. */
      handle_crc_error();
   }
}
```

## Data Structures

| | |
|---|---|
| struct | dmac_instance_ctrl_t |
| struct | dmac_extended_cfg_t |

## Macros

| | |
|---|---|
| #define | DMAC_MAX_NORMAL_TRANSFER_LENGTH |
| #define | DMAC_MAX_REPEAT_TRANSFER_LENGTH |
| #define | DMAC_MAX_BLOCK_TRANSFER_LENGTH |
| #define | DMAC_MAX_REPEAT_COUNT |
| #define | DMAC_MAX_BLOCK_COUNT |

## Data Structure Documentation

### ◆ dmac_instance_ctrl_t

| struct dmac_instance_ctrl_t |
| --- |
| Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open. |

### ◆ dmac_extended_cfg_t

| struct dmac_extended_cfg_t | |
| --- | --- |
| DMAC transfer configuration extension. This extension is required. | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel number, does not apply to all HAL drivers. |
| | |
| IRQn_Type | irq |
| | DMAC interrupt number. |
| | |
| uint8_t | ipl |
| | DMAC interrupt priority. |
| | |
| int32_t | offset |
| | Offset value used with transfer_addr_mode_t::TRANSFER_ADDR_MODE_OFFSET. |
| | |
| uint16_t | src_buffer_size |
| | |
| elc_event_t | activation_source |
| | |
| void(* | p_callback )(dmac_callback_args_t *cb_data) |
| | |
| void * | p_context |
| | |
| | |

## Field Documentation

### ◆ src_buffer_size

uint16_t dmac_extended_cfg_t::src_buffer_size

Source ring buffer size for TRANSFER_MODE_REPEAT_BLOCK.

### ◆ activation_source

elc_event_t dmac_extended_cfg_t::activation_source

Select which event will trigger the transfer.

***Note***

*Select ELC_EVENT_NONE for software activation in order to use softwareStart and softwareStart to trigger transfers.*

### ◆ p_callback

void(* dmac_extended_cfg_t::p_callback) (dmac_callback_args_t *cb_data)

Callback for transfer end interrupt.

### ◆ p_context

void* dmac_extended_cfg_t::p_context

Placeholder for user data. Passed to the user p_callback in transfer_callback_args_t.

## Macro Definition Documentation

### ◆ DMAC_MAX_NORMAL_TRANSFER_LENGTH

#define DMAC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in TRANSFER_MODE_NORMAL.

### ◆ DMAC_MAX_REPEAT_TRANSFER_LENGTH

#define DMAC_MAX_REPEAT_TRANSFER_LENGTH

Max number of transfers per repeat for TRANSFER_MODE_REPEAT.

### ◆ DMAC_MAX_BLOCK_TRANSFER_LENGTH

#define DMAC_MAX_BLOCK_TRANSFER_LENGTH

Max number of transfers per block in TRANSFER_MODE_BLOCK

### ◆ DMAC_MAX_REPEAT_COUNT

| #define DMAC_MAX_REPEAT_COUNT |
|---|
| Max configurable number of repeats to transfer in TRANSFER_MODE_REPEAT |

### ◆ DMAC_MAX_BLOCK_COUNT

| #define DMAC_MAX_BLOCK_COUNT |
|---|
| Max configurable number of blocks to transfer in TRANSFER_MODE_BLOCK |

## Function Documentation

### ◆ R_DMAC_Open()

fsp_err_t R_DMAC_Open ( transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg* )

Configure a DMAC channel.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_IP_CHANNEL_NOT_PRESENT | The configured channel is invalid. |
| FSP_ERR_IRQ_BSP_DISABLED | The IRQ associated with the activation source is not enabled in the BSP. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |

### ◆ R_DMAC_Reconfigure()

fsp_err_t R_DMAC_Reconfigure ( transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info* )

Reconfigure the transfer with new transfer info.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer is configured and will start when trigger occurs. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_ENABLED | DMAC is not enabled. The current configuration must not be valid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Reset()

fsp_err_t R_DMAC_Reset ( transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers* )

Reset transfer source, destination, and number of transfers.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer reset successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_ENABLED | DMAC is not enabled. The current configuration must not be valid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_SoftwareStart()

fsp_err_t R_DMAC_SoftwareStart ( transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode* )

If the mode is TRANSFER_START_MODE_SINGLE initiate a single transfer with software. If the mode is TRANSFER_START_MODE_REPEAT continue triggering transfers until all of the transfers are completed.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer started written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |
| FSP_ERR_UNSUPPORTED | Handle was not configured for software activation. |

### ◆ R_DMAC_SoftwareStop()

fsp_err_t R_DMAC_SoftwareStop ( transfer_ctrl_t *const *p_api_ctrl*)

Stop software transfers if they were started with TRANSFER_START_MODE_REPEAT.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer stopped written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Enable()

fsp_err_t R_DMAC_Enable ( transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers for the configured activation source.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_Disable()

fsp_err_t R_DMAC_Disable ( transfer_ctrl_t *const *p_api_ctrl*)

Disable transfers so that they are no longer triggered by the activation source.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Counter value written successfully. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

### ◆ R_DMAC_InfoGet()

fsp_err_t R_DMAC_InfoGet ( transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info* )

Set driver specific information in provided pointer.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Information has been written to p_info. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DMAC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DMAC_Reload()

fsp_err_t R_DMAC_Reload ( transfer_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *num_transfers* )

To update next transfer information without interruption during transfer.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This feature is not supported. |

### ◆ R_DMAC_CallbackSet()

fsp_err_t R_DMAC_CallbackSet ( transfer_ctrl_t *const *p_api_ctrl*, void(*)(dmac_callback_args_t *) *p_callback*, void *const *p_context*, dmac_callback_args_t *const *p_callback_memory* )

Updates the user callback with the option to provide memory for the callback argument structure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Callback updated successfully. |
| FSP_ERR_ASSERTION | A required pointer is NULL. |
| FSP_ERR_NOT_OPEN | The control block has not been opened. |

### ◆ R_DMAC_Close()

| fsp_err_t R_DMAC_Close ( transfer_ctrl_t *const *p_api_ctrl*) |
|---|
| Disable transfer and clean up internal data. Implements transfer_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Successful close.<br>FSP_ERR_ASSERTION — An input parameter is invalid.<br>FSP_ERR_NOT_OPEN — Handle is not initialized. Call R_DMAC_Open to initialize the control block. |

## 5.2.20.2 Transfer (r_dtc)

Modules » Transfer

### Functions

| | |
|---|---|
| fsp_err_t | R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg) |
| fsp_err_t | R_DTC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info) |
| fsp_err_t | R_DTC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers) |
| fsp_err_t | R_DTC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode) |
| fsp_err_t | R_DTC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl) |
| fsp_err_t | R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_properties) |
| fsp_err_t | R_DTC_Reload (transfer_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers) |
| fsp_err_t | R_DTC_CallbackSet (transfer_ctrl_t *const p_api_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory) |

fsp_err_t R_DTC_Close (transfer_ctrl_t *const p_api_ctrl)

### Detailed Description

Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface.

# Overview

The Data Transfer Controller (DTC) transfers data from one memory location to another without using the CPU.

The DTC uses a RAM based vector table. Each entry in the vector table corresponds to an entry in the ISR vector table. When the DTC is triggered by an interrupt, it reads the DTC vector table, fetches the transfer information, and then executes the transfer. After the transfer is executed, the DTC writes the updated transfer info back to the location pointed to by the DTC vector table.

### Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
- Chain transfers
- Address increment, decrement or fixed modes
- Can be triggered by any event that has reserved a slot in the interrupt vector table.
  - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

# Configuration

### Build Time Configurations for r_dtc

The following build time configurations are defined in fsp_cfg/r_dtc_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Parameter Checking | • Default (BSP)<br>• Enabled<br>• Disabled | Default (BSP) | If selected code for parameter checking is included in the build. |
| Linker section to keep DTC vector table | Manual Entry | .fsp_dtc_vector_table | Section to place the DTC vector table. |

### Configurations for Transfer > Transfer (r_dtc)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dtc).

| Configuration | Options | Default | Description |
|---|---|---|---|
| Name | Name must be a valid | g_transfer0 | Module name. |

| | C symbol | | |
|---|---|---|---|
| Mode | • Normal<br>• Repeat<br>• Block | Normal | Select the transfer mode.<br>Normal: One transfer per activation, transfer ends after Number of Transfers.<br>Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer repeats until stopped.<br>Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks. |
| Transfer Size | • 1 Byte<br>• 2 Bytes<br>• 4 Bytes | 2 Bytes | Select the transfer size. |
| Destination Address Mode | • Fixed<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the destination. |
| Source Address Mode | • Fixed<br>• Incremented<br>• Decremented | Fixed | Select the address mode for the source. |
| Repeat Area (Unused in Normal Mode) | • Destination<br>• Source | Source | Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode. |
| Interrupt Frequency | • After all transfers have completed<br>• After each transfer | After all transfers have completed | Select to have interrupt after each transfer or after last transfer. |
| Number of Transfers | Value must be a non-negative integer | 0 | Specify the number of transfers. |
| Number of Blocks (Valid only in Block Mode) | Must be a valid non-negative integer with a maximum configurable value of 65536. Applicable only in Block Mode. | 0 | Specify the number of blocks to transfer in Block mode. |

| | | | |
|---|---|---|---|
| Number of Transfer Descriptors | Value must be a non-negative integer | 1 | Specify the number of transfer descriptors. Users have to initialize descriptors if its value is greater than 1. |
| Activation Source | MCU Specific Options | | Select the DTC transfer start event. |

### Clock Configuration

The DTC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build or by using the CGC module at runtime.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### Source and Destination Configuration

R_DTC_Reset() API function should be called to set the Source and Destination address before starting the transfer operation.

### Transfer Modes

The DTC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time an interrupt is received by the DTC. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment or decrement to the next data unit after each transfer. A 16-bit counter (length) decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,256]. When the tranfer counter reaches 0, the counter is reset to its configured value and the repeat area (source or destination address) resets to its starting address and transfers will still be triggered by the interrupt.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,256]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

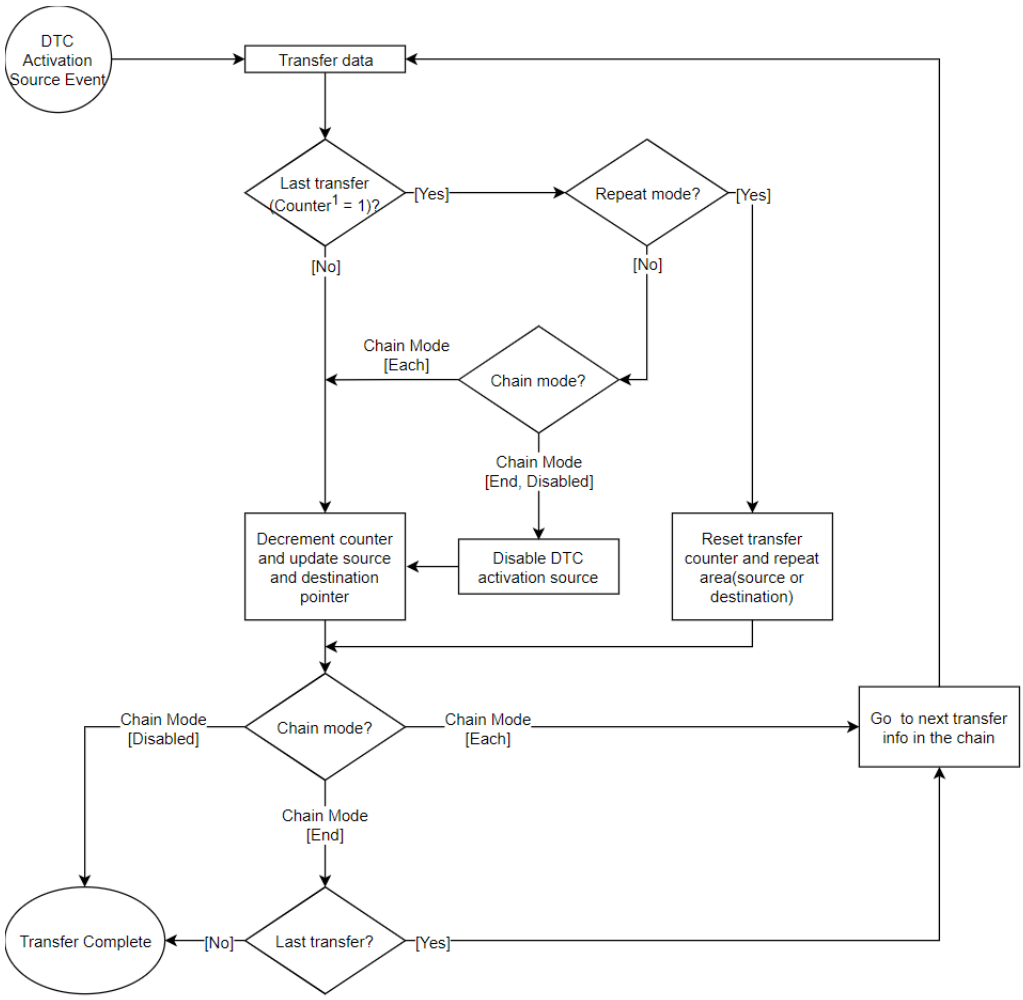
*Note*

*The source and destination address of the transfer must be aligned to the configured data unit.*

### Chaining Transfers

Multiple transfers can be configured for the same interrupt source by specifying an array of transfer_info_t structs instead of just passing a pointer to one. In this configuration, every transfer_info_t struct must be configured for a chain mode except for the last one. There are two types of chain mode; CHAIN_MODE_EACH and CHAIN_MODE_END. If a transfer is configured in

CHAIN_MODE_EACH then it triggers the next transfer in the chain after it completes each transfer. If a transfer is configured in CHAIN_MODE_END then it triggers the next transfer in the chain after it completes its last transfer.




1. **Counter** refers to transfer_info_t::**length** in normal and repeat mode and transfer_info_t::**num_blocks** in block mode.

Figure 348: DTC Transfer Flowchart

### Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

| | DTC | DMAC |
|---|---|---|
| Repeat Mode | • Repeats forever<br>• Max repeat size is 256 x 4 bytes | • Configurable number of repeats<br>• Max repeat size is 1024 x 4 bytes |
| Block Mode | • Max block size is 256 x 4 bytes | • Max block size is 1024 x 4 bytes |
| Channels | • One instance per interrupt | • MCU specific (8 channels or less) |

| | | |
|---|---|---|
| Chained Transfers | • Supported | • Not Supported |
| Software Trigger | • Must use the software ELC event | • Has support for software trigger without using software ELC event<br>• Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT |
| Offset Address Mode | • Not supported | • Supported |

**Additional Considerations**

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.
- The DTC interrupts the CPU using the activation source's IRQ. Each DMAC channel has its own IRQ.
- The necessary alignment of the transfer_info_t structs depends on the underlying MCU. The DTC_TRANSFER_INFO_ALIGNMENT macro is supplied to align the structs as necessary.

**Interrupts**

The DTC and DMAC interrupts behave differently. The DTC uses the configured event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

- **Normal Mode**

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | N/A |
| TRANSFER_IRQ_END | Interrupt after last transfer | Interrupt after last transfer |

- **Repeat Mode**

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each transfer | Interrupt after each repeat |
| TRANSFER_IRQ_END | Interrupt after each repeat | Interrupt after last transfer |

- **Block Mode**

| | DTC | DMAC |
|---|---|---|
| TRANSFER_IRQ_EACH | Interrupt after each block | Interrupt after each block |
| TRANSFER_IRQ_END | Interrupt after last block | Interrupt after last block |

*Note*

*DTC_VECTOR_TABLE_SIZE = (ICU_NVIC_IRQ_SOURCES x 4) Bytes*

### Peripheral Interrupts and DTC

When an interrupt is configured to trigger DTC transfers, the peripheral ISR will trigger on the following conditions:

- Each transfer completed (transfer_info_t::irq = TRANSFER_IRQ_EACH)
- Last transfer completed (transfer_info_t::irq = TRANSFER_IRQ_END)

For example, if SCI1_RXI is configured to trigger DTC transfers and a SCI1_RXI event occurs, the interrupt will not fire until the DTC transfer is completed. If the DTC transfer_info_t::irq is configured to only interrupt on the last transfer, then no RXI interrupts will occur until the last transfer is completed.

*Note*

*1. The DTC activation source must be enabled in the NVIC in order to trigger DTC transfers (Modules that are designed to integrate the R_DTC module will automatically handle this).*
*2. The DTC prioritizes activation sources by granting the smaller interrupt vector numbers higher priority. The priority of interrupts to the CPU is determined by the NVIC priority.*

### Low Power Modes

DTCST must be set to 0 before transitioning to any of the following:

- Module-stop state
- Software Standby mode without Snooze mode transition
- Deep Software Standby mode

*Note*

*1. R_LPM Module stops the DTC before entering deep software standby mode and software standby without snooze mode transition.*
*2. For more information see "Snooze Control Interface" and "Module-Stop Function" in the DTC section of the relevant hardware manual.*

### Limitations

Developers should be aware of the following limitations when using the DTC:

- If the DTC is configured to service many different activation sources, the system could run in to performance issues due to memory contention. To address this issue, it is recommended that the DTC vector table and transfer information be moved to their own dedicated memory area (Ex: SRAM0, SRAM1, SRAMHS). This allows memory accesses from different BUS Masters (CPU, DTC, DMAC, EDMAC and Graphics IPs) to occur in parallel.

## Examples

### Basic Example

This is a basic example of minimal use of the DTC in an application.

```
void dtc_minimal_example (void)

```

```
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the DTC to handle incoming transfer requests. */
    err = R_DTC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

## Data Structures

| | |
|---|---|
| struct | dtc_extended_cfg_t |
| struct | dtc_instance_ctrl_t |

## Macros

| | |
|---|---|
| #define | DTC_MAX_NORMAL_TRANSFER_LENGTH |
| #define | DTC_MAX_REPEAT_TRANSFER_LENGTH |
| #define | DTC_MAX_BLOCK_TRANSFER_LENGTH |
| #define | DTC_MAX_BLOCK_COUNT |
| #define | DTC_TRANSFER_INFO_ALIGNMENT |

## Data Structure Documentation

### ◆ dtc_extended_cfg_t

| struct dtc_extended_cfg_t | | |
|---|---|---|
| DTC transfer configuration extension. This extension is required. | | |
| Data Fields | | |
| IRQn_Type | activation_source | Select which IRQ will trigger the transfer. |

### ◆ dtc_instance_ctrl_t

| struct dtc_instance_ctrl_t |
|---|
| Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open. |

## Macro Definition Documentation

### ◆ DTC_MAX_NORMAL_TRANSFER_LENGTH

| #define DTC_MAX_NORMAL_TRANSFER_LENGTH |
| --- |
| Max configurable number of transfers in NORMAL MODE |

### ◆ DTC_MAX_REPEAT_TRANSFER_LENGTH

| #define DTC_MAX_REPEAT_TRANSFER_LENGTH |
| --- |
| Max number of transfers per repeat for REPEAT MODE |

### ◆ DTC_MAX_BLOCK_TRANSFER_LENGTH

| #define DTC_MAX_BLOCK_TRANSFER_LENGTH |
| --- |
| Max number of transfers per block in BLOCK MODE |

### ◆ DTC_MAX_BLOCK_COUNT

| #define DTC_MAX_BLOCK_COUNT |
| --- |
| Max configurable number of blocks to transfer in BLOCK MODE |

### ◆ DTC_TRANSFER_INFO_ALIGNMENT

| #define DTC_TRANSFER_INFO_ALIGNMENT |
| --- |
| Alignment required for transfer_info_t structures. |

## Function Documentation

### ◆ R_DTC_Open()

fsp_err_t R_DTC_Open ( transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg* )

Configure the vector table if it hasn't been configured, enable the Module and copy the pointer to the transfer info into the DTC vector table. Implements transfer_api_t::open.

Example:

```
/* Open the transfer instance with initial configuration. */
fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Successful open. Transfer transfer info pointer copied to DTC Vector table. Module started. DTC vector table configured. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_UNSUPPORTED | Address Mode Offset is selected. |
| FSP_ERR_ALREADY_OPEN | The control structure is already opened. |
| FSP_ERR_IN_USE | The index for this IRQ in the DTC vector table is already configured. |
| FSP_ERR_IRQ_BSP_DISABLED | The IRQ associated with the activation source is not enabled in the BSP. |

### ◆ R_DTC_Reconfigure()

fsp_err_t R_DTC_Reconfigure ( transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info* )

Copy pointer to transfer info into the DTC vector table and enable transfer in ICU. Implements transfer_api_t::reconfigure.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer is configured and will start when trigger occurs. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_NOT_ENABLED | Transfer source address is NULL or is not aligned correctly. Transfer destination address is NULL or is not aligned correctly. |

***Note***

*p_info must persist until all transfers are completed.*

### ◆ R_DTC_Reset()

fsp_err_t R_DTC_Reset ( transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers* )

Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfer reset successfully (transfers are enabled). |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_NOT_ENABLED | Transfer source address is NULL or is not aligned correctly. Transfer destination address is NULL or is not aligned correctly. |

### ◆ R_DTC_SoftwareStart()

fsp_err_t R_DTC_SoftwareStart ( transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode* )

Placeholder for unsupported softwareStart function. Implements transfer_api_t::softwareStart.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | DTC software start is not supported. |

### ◆ R_DTC_SoftwareStop()

fsp_err_t R_DTC_SoftwareStop ( transfer_ctrl_t *const *p_api_ctrl*)

Placeholder for unsupported softwareStop function. Implements transfer_api_t::softwareStop.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | DTC software stop is not supported. |

### ◆ R_DTC_Enable()

fsp_err_t R_DTC_Enable ( transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers on this activation source. Implements transfer_api_t::enable.

Example:

```
/* Enable the DTC to handle incoming transfer requests. */
   err = R_DTC_Enable(&g_transfer_ctrl);
   assert(FSP_SUCCESS == err);
```

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfers will be triggered by the activation source |
| FSP_ERR_ASSERTION | An input parameter is invalid. |
| FSP_ERR_UNSUPPORTED | Address Mode Offset is selected. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |

### ◆ R_DTC_Disable()

fsp_err_t R_DTC_Disable ( transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer on this activation source. Implements transfer_api_t::disable.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Transfers will not occur on activation events. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DTC_InfoGet()

fsp_err_t R_DTC_InfoGet ( transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_properties* )

Provides information about this transfer. Implements transfer_api_t::infoGet.

**Return values**

| | |
|---|---|
| FSP_SUCCESS | p_info updated with current instance information. |
| FSP_ERR_NOT_OPEN | Handle is not initialized. Call R_DTC_Open to initialize the control block. |
| FSP_ERR_ASSERTION | An input parameter is invalid. |

### ◆ R_DTC_Reload()

fsp_err_t R_DTC_Reload ( transfer_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *num_transfers* )

To update next transfer information without interruption during transfer.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | This feature is not supported. |

### ◆ R_DTC_CallbackSet()

fsp_err_t R_DTC_CallbackSet ( transfer_ctrl_t *const *p_api_ctrl*, void(*)(transfer_callback_args_t *) *p_callback*, void *const *p_context*, transfer_callback_args_t *const *p_callback_memory* )

Placeholder for unsupported callbackset function. Implements transfer_api_t::callbackSet.

**Return values**

| | |
|---|---|
| FSP_ERR_UNSUPPORTED | DTC does not support direct callbacks. |

◆ **R_DTC_Close()**

| fsp_err_t R_DTC_Close ( transfer_ctrl_t *const *p_api_ctrl*) |
|---|
| Disables DTC activation in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close.<br><br>**Return values**<br>FSP_SUCCESS — Successful close.<br>FSP_ERR_ASSERTION — An input parameter is invalid.<br>FSP_ERR_NOT_OPEN — Handle is not initialized. Call R_DTC_Open to initialize the control block. |

## 5.2.21 TrustZone

Modules

### Detailed Description

Arm TrustZone Modules.

### Modules

| | |
|---|---|
| Arm TrustZone Context RA Port (rm_tz_context) | RTOS Context Management for RA MCUs. |

### 5.2.21.1 Arm TrustZone Context RA Port (rm_tz_context)

Modules » TrustZone

RTOS Context Management for RA MCUs.

# Overview

Add this module to a secure TrustZone project to allow the associated non-secure project to use an RTOS. It is used by an RTOS port for RA MCUs (for example, the FreeRTOS Port (rm_freertos_port), which is automatically added to RA projects when FreeRTOS is selected during project creation).

*Note*

*The RTOS Context Management module does not provide any interfaces to the user. To use this module to port an*

*RTOS, consult the Arm documentation at https://arm-software.github.io/CMSIS_5/Core/html/group__context__trustzone__functions.html for further information.*

# Configuration

### Build Time Configurations for rm_tz_context

The following build time configurations are defined in fsp_cfg/rm_tz_context_cfg.h:

| Configuration | Options | Default | Description |
|---|---|---|---|
| Process Stack Slots | Value must be a non-negative integer greater than 0 | 8 | The maximum number of threads that can allocate a secure context. For applications using FreeRTOS, the Idle task requires 1 context as well. |
| Process Stack Size | Value must be a non-negative multiple of 8 | 256 | The maximum stack size of all non-secure callable functions. |

### Clock Configuration

This module does not use peripheral clocks.

### Pin Configuration

This module does not use I/O pins.

# Usage Notes

### TrustZone Integration

When using an RTOS in a TrustZone project, Arm recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the task has allocated a secure context. To allocate a secure context, reference the documentation for the RTOS port used. For example, reference TrustZone Integration when FreeRTOS is used.

### Sealing the Process Stack

This module seals each process stack by placing the value 0xFEF5EDA5 above the stack top. For more information, refer to section 3.5 "Sealing a Stack" in "Secure software guidelines for ARMv8-M": https://developer.arm.com/documentation/100720/0300.

# 5.3 Interfaces

## Detailed Description

FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer.

### Modules

| | |
|---|---|
| | Analog<br>Analog Interfaces. |
| | AI<br>AI Interfaces. |
| | Audio<br>Audio Interfaces. |
| | CapTouch<br>CapTouch Interfaces. |
| | Connectivity<br>Connectivity Interfaces. |
| | DSP<br>DSP Interfaces. |
| | Graphics<br>Graphics Interfaces. |
| | Input<br>Input Interfaces. |
| | Monitoring<br>Monitoring Interfaces. |
| | Motor<br>Motor Interfaces. |

| | |
|---|---|
| | Networking<br>Networking Interfaces. |
| | Power<br>Power Interfaces. |
| | Security<br>Security Interfaces. |
| | Sensor<br>Sensor Interfaces. |
| | Storage<br>Storage Interfaces. |
| | System<br>System Interfaces. |
| | Timers<br>Timers Interfaces. |
| | Transfer<br>Transfer Interfaces. |

## 5.3.1 Analog
Interfaces

### Detailed Description

Analog Interfaces.

### Modules

| | |
|---|---|
| | ADC Interface |

| | |
|---|---|
| | Interface for A/D Converters. |
| | Comparator Interface<br>Interface for comparators. |
| | DAC Interface<br>Interface for D/A converters. |
| | OPAMP Interface<br>Interface for Operational Amplifiers. |

#### 5.3.1.1 ADC Interface

Interfaces » Analog

**Detailed Description**

Interface for A/D Converters.

# Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

**Data Structures**

| | |
|---|---|
| struct | adc_status_t |
| struct | adc_callback_args_t |
| struct | adc_info_t |
| struct | adc_cfg_t |
| struct | adc_api_t |
| struct | adc_instance_t |

**Typedefs**

| | |
|---|---|
| typedef void | adc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | adc_mode_t |
| enum | adc_resolution_t |
| enum | adc_alignment_t |
| enum | adc_trigger_t |
| enum | adc_event_t |
| enum | adc_channel_t |
| enum | adc_group_id_t |
| enum | adc_group_mask_t |
| enum | adc_state_t |

## Data Structure Documentation

### ◆ adc_status_t

| struct adc_status_t | | |
|---|---|---|
| ADC status. | | |
| Data Fields | | |
| adc_state_t | state | Current state. |

### ◆ adc_callback_args_t

| struct adc_callback_args_t | | |
|---|---|---|
| ADC callback arguments definitions | | |
| Data Fields | | |
| uint16_t | unit | ADC device in use. |
| adc_event_t | event | ADC callback event. |
| void * | p_context | Placeholder for user data. |
| adc_channel_t | channel | Channel of conversion result. |
| uint64_t | channel_mask | Channel mask for conversion result. Only valid for r_adc_b and r_sdadc_b. |
| adc_group_mask_t | group_mask | Group Mask. |

### ◆ adc_info_t

| struct adc_info_t | | |
|---|---|---|
| ADC Information Structure for Transfer Interface | | |
| Data Fields | | |
| __I void * | p_address | The address to start reading the data from. |
| uint32_t | length | The total number of transfers to read. |
| transfer_size_t | transfer_size | The size of each transfer. |
| elc_peripheral_t | elc_peripheral | Name of the peripheral in the ELC list. |
| elc_event_t | elc_event | Name of the ELC event for the peripheral. |
| uint32_t | calibration_data | Temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage. |
| uint16_t | room_calibration_data | Room temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage. |
| uint16_t | low_calibration_data | Low temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage. |
| int16_t | slope_microvolts | Temperature sensor slope in microvolts/degrees C. |

### ◆ adc_cfg_t

| struct adc_cfg_t | |
|---|---|
| ADC general configuration | |
| **Data Fields** | |
| uint16_t | unit |
| | ADC unit to be used. |
| | |
| adc_mode_t | mode |
| | ADC operation mode. |
| | |
| adc_resolution_t | resolution |
| | ADC resolution. |

| | |
|---|---|
| | |
| adc_alignment_t | alignment |
| | Specify left or right alignment; ignored if addition used. |
| | |
| adc_trigger_t | trigger |
| | Default and Group A trigger source. |
| | |
| IRQn_Type | scan_end_irq |
| | Scan end IRQ number. |
| | |
| IRQn_Type | scan_end_b_irq |
| | Scan end group B IRQ number. |
| | |
| IRQn_Type | scan_end_c_irq |
| | Scan end group C IRQ number. |
| | |
| uint8_t | scan_end_ipl |
| | Scan end interrupt priority. |
| | |
| uint8_t | scan_end_b_ipl |
| | Scan end group B interrupt priority. |
| | |
| uint8_t | scan_end_c_ipl |
| | Scan end group C interrupt priority. |
| | |
| void(* | p_callback )(adc_callback_args_t *p_args) |
| | Callback function; set to NULL for none. |
| | |

| | |
|---|---|
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in adc_callback_args_t. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ adc_api_t

| struct adc_api_t | |
|---|---|
| ADC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanCfg )(adc_ctrl_t *const p_ctrl, void const *const p_extend) |
| | |
| fsp_err_t(* | scanStart )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | scanGroupStart )(adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask) |
| | |
| fsp_err_t(* | scanStop )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | scanStatusGet )(adc_ctrl_t *const p_ctrl, adc_status_t *p_status) |
| | |
| fsp_err_t(* | read )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data) |
| | |
| fsp_err_t(* | read32 )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data) |
| | |
| fsp_err_t(* | calibrate )(adc_ctrl_t *const p_ctrl, void const *p_extend) |
| | |
| fsp_err_t(* | offsetSet )(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset) |
| | |

| | |
|---|---|
| fsp_err_t(* | callbackSet )(adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(adc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.

**Precondition**

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_cfg | Pointer to configuration structure |

### ◆ scanCfg

fsp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, void const *const p_extend)

Configure the scan including the channels, groups, and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_extend | See implementation for details |

### ◆ scanStart

fsp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)

Start the scan (in case of a software trigger), or enable the hardware trigger.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ scanGroupStart

fsp_err_t(* adc_api_t::scanGroupStart) (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)

Start the scan group (in case of a software trigger), or enable the hardware trigger.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | group_mask | Mask of groups to start |

### ◆ scanStop

fsp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ scanStatusGet

fsp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl, adc_status_t *p_status)

Check scan status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_status | Pointer to store current status in |

### ◆ read

fsp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)

Read ADC conversion result.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | p_data | Pointer to variable to load value into. |

### ◆ read32

fsp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

Read ADC conversion result into a 32-bit word.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | p_data | Pointer to variable to load value into. |

### ◆ calibrate

fsp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void const *p_extend)

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_extend | Pointer to implementation specific arguments |

### ◆ offsetSet

fsp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | reg_id | ADC channel to read (see enumeration adc_channel_t) |
| [in] | offset | See implementation for details. |

### ◆ callbackSet

fsp_err_t(* adc_api_t::callbackSet) (adc_ctrl_t *const p_ctrl, void(*p_callback)(adc_callback_args_t *), void *const p_context, adc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the ADC control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)

Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ infoGet

| fsp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info) |
|---|

Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_adc_info | Pointer to ADC information structure |

### ◆ adc_instance_t

| struct adc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| adc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| adc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| void const * | p_channel_cfg | Pointer to the channel configuration structure for this instance. |
| adc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ adc_ctrl_t

| typedef void adc_ctrl_t |
|---|
| ADC control block. Allocate using driver instance control structure from driver instance header file. |

## Enumeration Type Documentation

### ◆ adc_mode_t

| enum adc_mode_t | |
|---|---|
| ADC operation mode definitions | |
| Enumerator | |
| ADC_MODE_SINGLE_SCAN | Single scan - one or more channels. |
| ADC_MODE_GROUP_SCAN | Two trigger sources to trigger scan for two groups which contain one or more channels. |
| ADC_MODE_CONTINUOUS_SCAN | Continuous scan - one or more channels. |

### ◆ adc_resolution_t

| enum adc_resolution_t | |
|---|---|
| ADC data resolution definitions | |
| Enumerator | |
| ADC_RESOLUTION_10_BIT | 10 bit resolution |
| ADC_RESOLUTION_8_BIT | 8 bit resolution |
| ADC_RESOLUTION_12_BIT | 12 bit resolution |
| ADC_RESOLUTION_10_BIT | 10 bit resolution |
| ADC_RESOLUTION_8_BIT | 8 bit resolution |
| ADC_RESOLUTION_12_BIT | 12 bit resolution |
| ADC_RESOLUTION_10_BIT | 10 bit resolution |
| ADC_RESOLUTION_8_BIT | 8 bit resolution |
| ADC_RESOLUTION_12_BIT | 12 bit resolution |
| ADC_RESOLUTION_12_BIT | 12 bit resolution |
| ADC_RESOLUTION_10_BIT | 10 bit resolution |
| ADC_RESOLUTION_8_BIT | 8 bit resolution |
| ADC_RESOLUTION_14_BIT | 14 bit resolution |
| ADC_RESOLUTION_16_BIT | 16 bit resolution |
| ADC_RESOLUTION_24_BIT | 24 bit resolution |

### ◆ adc_alignment_t

| enum adc_alignment_t | |
|---|---|
| ADC data alignment definitions | |
| Enumerator | |
| ADC_ALIGNMENT_RIGHT | Data alignment right. |
| ADC_ALIGNMENT_LEFT | Data alignment left. |

### ◆ adc_trigger_t

| enum adc_trigger_t | |
|---|---|
| ADC trigger mode definitions | |
| Enumerator | |
| ADC_TRIGGER_SOFTWARE | Software trigger; not for group modes. |
| ADC_TRIGGER_SYNC_ELC | Synchronous trigger via ELC. |
| ADC_TRIGGER_ASYNC_EXTERNAL | External asynchronous trigger; not for group modes. |

### ◆ adc_event_t

| enum adc_event_t | |
|---|---|
| ADC callback event definitions | |
| Enumerator | |
| ADC_EVENT_SCAN_COMPLETE | Normal/Group A scan complete. |
| ADC_EVENT_SCAN_COMPLETE_GROUP_B | Group B scan complete. |
| ADC_EVENT_SCAN_COMPLETE_GROUP_C | Group C scan complete. |
| ADC_EVENT_CALIBRATION_COMPLETE | Calibration complete. |
| ADC_EVENT_CONVERSION_COMPLETE | Conversion complete. |
| ADC_EVENT_CALIBRATION_REQUEST | Calibration requested. |
| ADC_EVENT_CONVERSION_ERROR | Scan error. |
| ADC_EVENT_OVERFLOW | Overflow occurred. |
| ADC_EVENT_LIMIT_CLIP | Limiter clipping occurred. |
| ADC_EVENT_FIFO_READ_REQUEST | FIFO read requested. |
| ADC_EVENT_FIFO_OVERFLOW | FIFO overflow occurred. |
| ADC_EVENT_WINDOW_COMPARE_A | Window A comparison condition met. |
| ADC_EVENT_WINDOW_COMPARE_B | Window B comparison condition met. |
| ADC_EVENT_ZERO_CROSS_DETECTION | Zero-cross detection interrupt. |
| ADC_EVENT_CAPTURE_A | Capture A conversion complete. |
| ADC_EVENT_CAPTURE_B | Capture B conversion complete. |

### ◆ adc_channel_t

| enum adc_channel_t | |
|---|---|
| ADC channels | |
| Enumerator | |
| ADC_CHANNEL_0 | Channel 0 for select mode or channel 0 to 3 for scan mode. |
| ADC_CHANNEL_1 | Channel 1 for select mode or channel 1 to 4 for scan mode. |
| ADC_CHANNEL_2 | Channel 2 for select mode or channel 2 to 5 for scan mode. |
| ADC_CHANNEL_3 | Channel 3 for select mode or channel 3 to 6 for scan mode. |
| ADC_CHANNEL_4 | Channel 4 for select mode or channel 4 to 7 for scan mode. |
| ADC_CHANNEL_5 | Channel 5 for select mode. |
| ADC_CHANNEL_6 | Channel 6 for select mode. |
| ADC_CHANNEL_7 | Channel 7 for select mode. |
| ADC_CHANNEL_21 | Channel 21 for select mode. |
| ADC_CHANNEL_22 | Channel 22 for select mode. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output voltage for select mode. |
| ADC_CHANNEL_VOLT | Internal reference voltage for select mode. |
| ADC_CHANNEL_POSITIVE_SIDE_VREF | Select positive reference voltage as target conversion. |
| ADC_CHANNEL_NEGATIVE_SIDE_VREF | Select negative reference voltage as target conversion. |
| ADC_CHANNEL_0 | Channel 0 for select mode or channel 0 to 3 for scan mode. |
| ADC_CHANNEL_1 | Channel 1 for select mode or channel 1 to 4 for scan mode. |
| | |

| | |
|---|---|
| ADC_CHANNEL_2 | Channel 2 for select mode or channel 2 to 5 for scan mode. |
| ADC_CHANNEL_3 | Channel 3 for select mode or channel 3 to 6 for scan mode. |
| ADC_CHANNEL_4 | Channel 4 for select mode or channel 4 to 7 for scan mode. |
| ADC_CHANNEL_5 | Channel 5 for select mode or channel 5 to 8 for scan mode. |
| ADC_CHANNEL_6 | Channel 6 for select mode or channel 6 to 9 for scan mode. |
| ADC_CHANNEL_7 | Channel 7 for select mode or channel 7 to 10 for scan mode. |
| ADC_CHANNEL_8 | Channel 8 for select mode or channel 8 to 11 for scan mode. |
| ADC_CHANNEL_9 | Channel 9 for select mode or channel 9 to 12 for scan mode. |
| ADC_CHANNEL_10 | Channel 10 for select mode. |
| ADC_CHANNEL_11 | Channel 11 for select mode. |
| ADC_CHANNEL_12 | Channel 12 for select mode. |
| ADC_CHANNEL_21 | Channel 21 for select mode. |
| ADC_CHANNEL_22 | Channel 22 for select mode. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output voltage for select mode. |
| ADC_CHANNEL_VOLT | Internal reference voltage for select mode. |
| ADC_CHANNEL_POSITIVE_SIDE_VREF | Select positive reference voltage as target conversion. |
| ADC_CHANNEL_NEGATIVE_SIDE_VREF | Select negative reference voltage as target conversion. |
| ADC_CHANNEL_0 | Channel 0 for select mode or channel 0 to 3 for scan mode. |
| ADC_CHANNEL_1 | Channel 1 for select mode or channel 1 to 4 for scan mode. |

| | |
|---|---|
| ADC_CHANNEL_2 | Channel 2 for select mode or channel 2 to 5 for scan mode. |
| ADC_CHANNEL_3 | Channel 3 for select mode or channel 3 to 6 for scan mode. |
| ADC_CHANNEL_4 | Channel 4 for select mode or channel 4 to 7 for scan mode. |
| ADC_CHANNEL_5 | Channel 5 for select mode or channel 5 to 8 for scan mode. |
| ADC_CHANNEL_6 | Channel 6 for select mode or channel 6 to 9 for scan mode. |
| ADC_CHANNEL_7 | Channel 7 for select mode or channel 7 to 10 for scan mode. |
| ADC_CHANNEL_8 | Channel 8 for select mode or channel 8 to 11 for scan mode. |
| ADC_CHANNEL_9 | Channel 9 for select mode or channel 9 to 12 for scan mode. |
| ADC_CHANNEL_10 | Channel 10 for select mode. |
| ADC_CHANNEL_TSCAP_VOLT | CTSU TSCAP voltage for select mode. |
| ADC_CHANNEL_21 | Channel 21 for select mode. |
| ADC_CHANNEL_22 | Channel 22 for select mode. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output voltage for select mode. |
| ADC_CHANNEL_VOLT | Internal reference voltage for select mode. |
| ADC_CHANNEL_POSITIVE_SIDE_VREF | Select positive reference voltage as target conversion. |
| ADC_CHANNEL_NEGATIVE_SIDE_VREF | Select negative reference voltage as target conversion. |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| | |

| | |
|---|---|
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_23 | ADC channel 23. |
| ADC_CHANNEL_24 | ADC channel 24. |
| ADC_CHANNEL_25 | ADC channel 25. |
| ADC_CHANNEL_26 | ADC channel 26. |
| | |

| | |
|---|---|
| ADC_CHANNEL_27 | ADC channel 27. |
| ADC_CHANNEL_28 | ADC channel 28. |
| ADC_CHANNEL_SELF_DIAGNOSIS | Self-Diagnosis channel. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |
| ADC_CHANNEL_DA0 | D/A Converter Channel 0. |
| ADC_CHANNEL_DA1 | D/A Converter Channel 1. |
| ADC_CHANNEL_DA2 | D/A Converter Channel 2. |
| ADC_CHANNEL_DA3 | D/A Converter Channel 3. |
| ADC_CHANNEL_FIRST_INTERNAL | Highest physical channel. |
| ADC_CHANNEL_LAST_INTERNAL | First internal channel. |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| | |

| | |
|---|---|
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC0 | Self-Diagnosis channel for ADC Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC1 | Self-Diagnosis channel for ADC Unit 1. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |
| ADC_CHANNEL_VBATT | VBATT 1/3 voltage monitor. |
| ADC_CHANNEL_DA0 | D/A converter channel 0. |
| ADC_CHANNEL_DA1 | D/A converter channel 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH0 | Self-Diagnosis Sample-and-hold Circuit Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH1 | Self-Diagnosis Sample-and-hold Circuit Unit 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH2 | Self-Diagnosis Sample-and-hold Circuit Unit 2. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH4 | Self-Diagnosis Sample-and-hold Circuit Unit 4. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH5 | Self-Diagnosis Sample-and-hold Circuit Unit 5. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH6 | Self-Diagnosis Sample-and-hold Circuit Unit 6. |
| ADC_CHANNEL_FIRST_INTERNAL | Highest physical channel. |
| | |

| | |
|---|---|
| ADC_CHANNEL_LAST_INTERNAL | First internal channel. |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| | |

| | |
|---|---|
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC0 | Self-Diagnosis channel for ADC Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC1 | Self-Diagnosis channel for ADC Unit 1. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |
| ADC_CHANNEL_VBATT | VBATT 1/3 voltage monitor. |
| ADC_CHANNEL_DA0 | D/A converter channel 0. |
| ADC_CHANNEL_DA1 | D/A converter channel 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH0 | Self-Diagnosis Sample-and-hold Circuit Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH1 | Self-Diagnosis Sample-and-hold Circuit Unit 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH2 | Self-Diagnosis Sample-and-hold Circuit Unit 2. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH4 | Self-Diagnosis Sample-and-hold Circuit Unit 4. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH5 | Self-Diagnosis Sample-and-hold Circuit Unit 5. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH6 | Self-Diagnosis Sample-and-hold Circuit Unit 6. |
| ADC_CHANNEL_FIRST_INTERNAL | Highest physical channel. |
| ADC_CHANNEL_LAST_INTERNAL | First internal channel. |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| | |

| | |
|---|---|
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC0 | Self-Diagnosis channel for ADC Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC1 | Self-Diagnosis channel for ADC Unit 1. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |
| ADC_CHANNEL_VBATT | VBATT 1/3 voltage monitor. |
| ADC_CHANNEL_DA0 | D/A converter channel 0. |
| ADC_CHANNEL_DA1 | D/A converter channel 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH0 | Self-Diagnosis Sample-and-hold Circuit Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH1 | Self-Diagnosis Sample-and-hold Circuit Unit 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH2 | Self-Diagnosis Sample-and-hold Circuit Unit 2. |
| | |

| | |
|---|---|
| ADC_CHANNEL_SELF_DIAGNOSIS_SH4 | Self-Diagnosis Sample-and-hold Circuit Unit 4. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH5 | Self-Diagnosis Sample-and-hold Circuit Unit 5. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH6 | Self-Diagnosis Sample-and-hold Circuit Unit 6. |
| ADC_CHANNEL_FIRST_INTERNAL | Highest physical channel. |
| ADC_CHANNEL_LAST_INTERNAL | First internal channel. |
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| | |

| | |
|---|---|
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC0 | Self-Diagnosis channel for ADC Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_ADC1 | Self-Diagnosis channel for ADC Unit 1. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |
| ADC_CHANNEL_VBATT | VBATT 1/3 voltage monitor. |
| ADC_CHANNEL_DA0 | D/A converter channel 0. |
| ADC_CHANNEL_DA1 | D/A converter channel 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH0 | Self-Diagnosis Sample-and-hold Circuit Unit 0. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH1 | Self-Diagnosis Sample-and-hold Circuit Unit 1. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH2 | Self-Diagnosis Sample-and-hold Circuit Unit 2. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH4 | Self-Diagnosis Sample-and-hold Circuit Unit 4. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH5 | Self-Diagnosis Sample-and-hold Circuit Unit 5. |
| ADC_CHANNEL_SELF_DIAGNOSIS_SH6 | Self-Diagnosis Sample-and-hold Circuit Unit 6. |
| ADC_CHANNEL_FIRST_INTERNAL | Highest physical channel. |
| ADC_CHANNEL_LAST_INTERNAL | First internal channel. |
| ADC_CHANNEL_CAPTURE_A_MASK | Last internal channel.<br>Capture A channel mask |
| ADC_CHANNEL_CAPTURE_B_MASK | Capture B channel mask. |
| ADC_CHANNEL_OVERCURRENT_MASK | Overcurrent channel mask. |
| ADC_CHANNEL_OVERCURRENT0_CAPTURE_MASK | Overcurrent0 capture channel mask. |
| ADC_CHANNEL_OVERCURRENT1_CAPTURE_MASK | |

| | |
|---|---|
| | Overcurrent1 capture channel mask. |
| ADC_CHANNEL_OVERCURRENT2_CAPTURE_MASK | Overcurrent2 capture channel mask. |
| ADC_CHANNEL_OVERCURRENT_SUM_CAPTURE_MASK | Overcurrent sum capture channel mask. |
| ADC_CHANNEL_0_CAPTURE_A | Channel 0 - Capture A channel. |
| ADC_CHANNEL_0_CAPTURE_B | Channel 0 - Capture B channel. |
| ADC_CHANNEL_0_OVERCURRENT | Channel 0 - Overcurrent channel. |
| ADC_CHANNEL_0_OVERCURRENT0_CAPTURE | Channel 0 - Overcurrent0 capture channel. |
| ADC_CHANNEL_0_OVERCURRENT1_CAPTURE | Channel 0 - Overcurrent1 capture channel. |
| ADC_CHANNEL_0_OVERCURRENT2_CAPTURE | Channel 0 - Overcurrent2 capture channel. |
| ADC_CHANNEL_0_OVERCURRENT_SUM_CAPTURE | Channel 0 - Overcurrent sum capture channel. |
| ADC_CHANNEL_1_CAPTURE_A | Channel 1 - Capture A channel. |
| ADC_CHANNEL_1_CAPTURE_B | Channel 1 - Capture B channel. |
| ADC_CHANNEL_1_OVERCURRENT | Channel 1 - Overcurrent channel. |
| ADC_CHANNEL_1_OVERCURRENT0_CAPTURE | Channel 1 - Overcurrent0 capture channel. |
| ADC_CHANNEL_1_OVERCURRENT1_CAPTURE | Channel 1 - Overcurrent1 capture channel. |
| ADC_CHANNEL_1_OVERCURRENT2_CAPTURE | Channel 1 - Overcurrent2 capture channel. |
| ADC_CHANNEL_1_OVERCURRENT_SUM_CAPTURE | Channel 1 - Overcurrent sum capture channel. |
| ADC_CHANNEL_2_CAPTURE_A | Channel 2 - Capture A channel. |
| ADC_CHANNEL_2_CAPTURE_B | Channel 2 - Capture B channel. |
| ADC_CHANNEL_2_OVERCURRENT | Channel 2 - Overcurrent channel. |
| ADC_CHANNEL_2_OVERCURRENT0_CAPTURE | Channel 2 - Overcurrent0 capture channel. |
| ADC_CHANNEL_2_OVERCURRENT1_CAPTURE | Channel 2 - Overcurrent1 capture channel. |
| ADC_CHANNEL_2_OVERCURRENT2_CAPTURE | Channel 2 - Overcurrent2 capture channel. |
| ADC_CHANNEL_2_OVERCURRENT_SUM_CAPTURE | Channel 2 - Overcurrent sum capture channel. |

| | |
|---|---|
| ADC_CHANNEL_0 | ADC channel 0. |
| ADC_CHANNEL_1 | ADC channel 1. |
| ADC_CHANNEL_2 | ADC channel 2. |
| ADC_CHANNEL_3 | ADC channel 3. |
| ADC_CHANNEL_4 | ADC channel 4. |
| ADC_CHANNEL_5 | ADC channel 5. |
| ADC_CHANNEL_6 | ADC channel 6. |
| ADC_CHANNEL_7 | ADC channel 7. |
| ADC_CHANNEL_8 | ADC channel 8. |
| ADC_CHANNEL_9 | ADC channel 9. |
| ADC_CHANNEL_10 | ADC channel 10. |
| ADC_CHANNEL_11 | ADC channel 11. |
| ADC_CHANNEL_12 | ADC channel 12. |
| ADC_CHANNEL_13 | ADC channel 13. |
| ADC_CHANNEL_14 | ADC channel 14. |
| ADC_CHANNEL_15 | ADC channel 15. |
| ADC_CHANNEL_16 | ADC channel 16. |
| ADC_CHANNEL_17 | ADC channel 17. |
| ADC_CHANNEL_18 | ADC channel 18. |
| ADC_CHANNEL_19 | ADC channel 19. |
| ADC_CHANNEL_20 | ADC channel 20. |
| ADC_CHANNEL_21 | ADC channel 21. |
| ADC_CHANNEL_22 | ADC channel 22. |
| ADC_CHANNEL_23 | ADC channel 23. |

| | |
|---|---|
| ADC_CHANNEL_24 | ADC channel 24. |
| ADC_CHANNEL_25 | ADC channel 25. |
| ADC_CHANNEL_26 | ADC channel 26. |
| ADC_CHANNEL_27 | ADC channel 27. |
| ADC_CHANNEL_28 | ADC channel 28. |
| ADC_CHANNEL_DUPLEX_A | Data duplexing register A. |
| ADC_CHANNEL_DUPLEX_B | Data duplexing register B. |
| ADC_CHANNEL_DUPLEX | Data duplexing register. |
| ADC_CHANNEL_TEMPERATURE | Temperature sensor output. |
| ADC_CHANNEL_VOLT | Internal reference voltage. |

### ◆ adc_group_id_t

| enum adc_group_id_t | |
|---|---|
| Enumerator | |
| ADC_GROUP_ID_0 | Group ID 0. |
| ADC_GROUP_ID_1 | Group ID 1. |
| ADC_GROUP_ID_2 | Group ID 2. |
| ADC_GROUP_ID_3 | Group ID 3. |
| ADC_GROUP_ID_4 | Group ID 4. |
| ADC_GROUP_ID_5 | Group ID 5. |
| ADC_GROUP_ID_6 | Group ID 6. |
| ADC_GROUP_ID_7 | Group ID 7. |
| ADC_GROUP_ID_8 | Group ID 8. |

### ◆ adc_group_mask_t

| enum adc_group_mask_t | |
|---|---|
| Enumerator | |
| ADC_GROUP_MASK_NONE | Group Mask Unknown or None. |
| ADC_GROUP_MASK_0 | Group Mask 0. |
| ADC_GROUP_MASK_1 | Group Mask 1. |
| ADC_GROUP_MASK_2 | Group Mask 2. |
| ADC_GROUP_MASK_3 | Group Mask 3. |
| ADC_GROUP_MASK_4 | Group Mask 4. |
| ADC_GROUP_MASK_5 | Group Mask 5. |
| ADC_GROUP_MASK_6 | Group Mask 6. |
| ADC_GROUP_MASK_7 | Group Mask 7. |
| ADC_GROUP_MASK_8 | Group Mask 8. |
| ADC_GROUP_MASK_ALL | All Groups. |

### ◆ adc_state_t

| enum adc_state_t | |
|---|---|
| ADC states. | |
| Enumerator | |
| ADC_STATE_IDLE | ADC is idle. |
| ADC_STATE_SCAN_IN_PROGRESS | ADC scan in progress. |
| ADC_STATE_CALIBRATION_IN_PROGRESS | ADC calibration in progress - Not used by all ADC instances. |

## 5.3.1.2 Comparator Interface

Interfaces » Analog

### Detailed Description

Interface for comparators.

# Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

### Data Structures

| | |
|---|---|
| struct | comparator_info_t |
| struct | comparator_status_t |
| struct | comparator_callback_args_t |
| struct | comparator_cfg_t |
| struct | comparator_api_t |
| struct | comparator_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | comparator_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | comparator_mode_t |
| enum | comparator_trigger_t |
| enum | comparator_polarity_invert_t |
| enum | comparator_pin_output_t |
| enum | comparator_filter_t |
| enum | comparator_state_t |

### Data Structure Documentation

#### ◆ comparator_info_t

| struct comparator_info_t |
|---|
| Comparator information. |
| Data Fields |

| | | |
|---|---|---|
| uint32_t | min_stabilization_wait_us | Minimum stabilization wait time in microseconds. |

### ◆ comparator_status_t

| struct comparator_status_t | | |
|---|---|---|
| Comparator status. | | |
| Data Fields | | |
| comparator_state_t | state | Current comparator state. |

### ◆ comparator_callback_args_t

| struct comparator_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in comparator_api_t::open function in comparator_cfg_t. |
| uint32_t | channel | The physical hardware channel that caused the interrupt. |

### ◆ comparator_cfg_t

| struct comparator_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint8_t | channel |
| | Hardware channel used. |
| | |
| comparator_mode_t | mode |
| | Normal or window mode. |
| | |
| comparator_trigger_t | trigger |
| | Trigger setting. |
| | |
| comparator_filter_t | filter |
| | Digital filter clock divisor setting. |
| | |

| | |
|---|---|
| comparator_polarity_invert_t | invert |
| | Whether to invert output. |
| | |
| comparator_pin_output_t | pin_output |
| | Whether to include output on output pin. |
| | |
| uint8_t | vref_select |
| | Internal Vref Select. |
| | |
| uint8_t | ipl |
| | Interrupt priority. |
| | |
| IRQn_Type | irq |
| | NVIC interrupt number. |
| | |
| void(* | p_callback )(comparator_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Comparator hardware dependent configuration. |
| | |

## Field Documentation

### ◆ p_callback

void(* comparator_cfg_t::p_callback) (comparator_callback_args_t *p_args)

Callback called when comparator event occurs.

### ◆ p_context

| void* comparator_cfg_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in comparator_callback_args_t. |

### ◆ comparator_api_t

| struct comparator_api_t | |
|---|---|
| Comparator functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | outputEnable )(comparator_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(comparator_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

| fsp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg) |
|---|
| Initialize the comparator. |

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ outputEnable

fsp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)

Start the comparator.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ infoGet

fsp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_info | Comparator information stored here |

### ◆ statusGet

fsp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)

Provide current comparator status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_status | Status stored here |

### ◆ close

fsp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)

Stop the comparator.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ comparator_instance_t

struct comparator_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| comparator_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| comparator_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| comparator_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ comparator_ctrl_t

typedef void comparator_ctrl_t

Includes board and MCU related header files. Comparator control block. Allocate an instance specific control block to pass into the comparator API calls.

## Enumeration Type Documentation

### ◆ comparator_mode_t

enum comparator_mode_t

Select whether to invert the polarity of the comparator output.

| Enumerator | |
|---|---|
| COMPARATOR_MODE_NORMAL | Normal mode. |
| COMPARATOR_MODE_WINDOW | Window mode, not supported by all implementations. |

### ◆ comparator_trigger_t

enum comparator_trigger_t

Trigger type: rising edge, falling edge, both edges, low level.

| Enumerator | |
|---|---|
| COMPARATOR_TRIGGER_RISING | Rising edge trigger. |
| COMPARATOR_TRIGGER_FALLING | Falling edge trigger. |
| COMPARATOR_TRIGGER_BOTH_EDGE | Both edges trigger. |

### ◆ comparator_polarity_invert_t

| enum comparator_polarity_invert_t | |
|---|---|
| Select whether to invert the polarity of the comparator output. | |
| Enumerator | |
| COMPARATOR_POLARITY_INVERT_OFF | Do not invert polarity. |
| COMPARATOR_POLARITY_INVERT_ON | Invert polarity. |

### ◆ comparator_pin_output_t

| enum comparator_pin_output_t | |
|---|---|
| Select whether to include the comparator output on the output pin. | |
| Enumerator | |
| COMPARATOR_PIN_OUTPUT_OFF | Do not include comparator output on output pin. |
| COMPARATOR_PIN_OUTPUT_ON | Include comparator output on output pin. |

### ◆ comparator_filter_t

| enum comparator_filter_t | |
|---|---|
| Comparator digital filtering sample clock divisor settings. | |
| Enumerator | |
| COMPARATOR_FILTER_OFF | Disable debounce filter. |
| COMPARATOR_FILTER_1 | Filter using PCLK divided by 1, not supported by all implementations. |
| COMPARATOR_FILTER_8 | Filter using PCLK divided by 8. |
| COMPARATOR_FILTER_16 | Filter using PCLK divided by 16, not supported by all implementations. |
| COMPARATOR_FILTER_32 | Filter using PCLK divided by 32. |

### ◆ comparator_state_t

| enum comparator_state_t | |
|---|---|
| Current comparator state. | |
| Enumerator | |
| COMPARATOR_STATE_OUTPUT_LOW | VCMP < VREF if polarity is not inverted, VCMP > VREF if inverted. |
| COMPARATOR_STATE_OUTPUT_HIGH | VCMP > VREF if polarity is not inverted, VCMP < VREF if inverted. |
| COMPARATOR_STATE_OUTPUT_DISABLED | comparator_api_t::outputEnable() has not been called |

#### 5.3.1.3 DAC Interface

Interfaces » Analog

#### Detailed Description

Interface for D/A converters.

# Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

**Data Structures**

| | |
|---|---|
| struct | dac_info_t |
| struct | dac_cfg_t |
| struct | dac_api_t |
| struct | dac_instance_t |

**Typedefs**

| | |
|---|---|
| typedef void | dac_ctrl_t |

**Enumerations**

| | |
|---|---|
| enum | dac_data_format_t |

## Data Structure Documentation

### ◆ dac_info_t

| struct dac_info_t | | |
|---|---|---|
| DAC information structure to store various information for a DAC | | |
| Data Fields | | |
| uint8_t | bit_width | Resolution of the DAC. |

### ◆ dac_cfg_t

| struct dac_cfg_t | | |
|---|---|---|
| DAC Open API configuration parameter | | |
| Data Fields | | |
| uint8_t | channel | ID associated with this DAC channel. |
| bool | ad_da_synchronized | AD/DA synchronization. |
| void const * | p_extend | |

### ◆ dac_api_t

| struct dac_api_t | |
|---|---|
| DAC driver structure. General DAC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(dac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | write )(dac_ctrl_t *const p_ctrl, uint16_t value) |
| | |
| fsp_err_t(* | start )(dac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(dac_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* dac_api_t::open) (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* dac_api_t::close) (dac_ctrl_t *const p_ctrl)

Close the D/A Converter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ write

fsp_err_t(* dac_api_t::write) (dac_ctrl_t *const p_ctrl, uint16_t value)

Write sample value to the D/A Converter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |
| [in] | value | Sample value to be written to the D/A Converter. |

### ◆ start

| fsp_err_t(* dac_api_t::start) (dac_ctrl_t *const p_ctrl) |
|---|

Start the D/A Converter if it has not been started yet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ stop

| fsp_err_t(* dac_api_t::stop) (dac_ctrl_t *const p_ctrl) |
|---|

Stop the D/A Converter if the converter is running.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in dac_api_t::open call for this timer. |

### ◆ dac_instance_t

| struct dac_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| dac_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| dac_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| dac_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ dac_ctrl_t

| typedef void dac_ctrl_t |
|---|
| DAC control block. Allocate an instance specific control block to pass into the DAC API calls. |

## Enumeration Type Documentation

### ◆ dac_data_format_t

| enum dac_data_format_t | |
|---|---|
| DAC Open API data format settings. | |
| Enumerator | |
| DAC_DATA_FORMAT_FLUSH_RIGHT | LSB of data is flush to the right leaving the top 4 bits unused. |
| DAC_DATA_FORMAT_FLUSH_LEFT | MSB of data is flush to the left leaving the bottom 4 bits unused. |

### 5.3.1.4 OPAMP Interface

Interfaces » Analog

#### Detailed Description

Interface for Operational Amplifiers.

## Summary

The OPAMP interface provides standard operational amplifier functionality, including starting and stopping the amplifier.

#### Data Structures

| | |
|---|---|
| struct | opamp_trim_args_t |
| struct | opamp_info_t |
| struct | opamp_status_t |
| struct | opamp_cfg_t |
| struct | opamp_api_t |
| struct | opamp_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | opamp_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | opamp_trim_cmd_t |

| | |
|---|---|
| enum | opamp_trim_input_t |

## Data Structure Documentation

### ◆ opamp_trim_args_t

| struct opamp_trim_args_t | | |
|---|---|---|
| OPAMP trim arguments. | | |
| Data Fields | | |
| uint8_t | channel | Channel. |
| opamp_trim_input_t | input | Which input of the channel above. |

### ◆ opamp_info_t

| struct opamp_info_t | | |
|---|---|---|
| OPAMP information. | | |
| Data Fields | | |
| uint32_t | min_stabilization_wait_us | Minimum stabilization wait time in microseconds. |

### ◆ opamp_status_t

| struct opamp_status_t | | |
|---|---|---|
| OPAMP status. | | |
| Data Fields | | |
| uint32_t | operating_channel_mask | Bitmask of channels currently operating. |

### ◆ opamp_cfg_t

| struct opamp_cfg_t | | |
|---|---|---|
| OPAMP general configuration. | | |
| Data Fields | | |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ opamp_api_t

| struct opamp_api_t | |
|---|---|
| OPAMP functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg) |
| | |

| | |
|---|---|
| fsp_err_t(* | start )(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask) |
| | |
| fsp_err_t(* | stop )(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask) |
| | |
| fsp_err_t(* | trim )(opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args) |
| | |
| fsp_err_t(* | infoGet )(opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(opamp_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)

Initialize the operational amplifier.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ start

fsp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Start the op-amp(s).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | channel_mask | Bitmask of channels to start |

### ◆ stop

fsp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Stop the op-amp(s).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | channel_mask | Bitmask of channels to stop |

### ◆ trim

fsp_err_t(* opamp_api_t::trim) (opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)

Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | cmd | Trim command |
| [in] | p_args | Pointer to arguments for the command |

### ◆ infoGet

fsp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_info | OPAMP information stored here |

### ◆ statusGet

| fsp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status) |
|---|

Provide status of each op-amp channel.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_status | Status stored here |

### ◆ close

| fsp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl) |
|---|

Close the specified OPAMP unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ opamp_instance_t

| struct opamp_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| opamp_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| opamp_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| opamp_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ opamp_ctrl_t

| typedef void opamp_ctrl_t |
|---|
| OPAMP control block. Allocate using driver instance control structure from driver instance header file. |

## Enumeration Type Documentation

### ◆ opamp_trim_cmd_t

| enum opamp_trim_cmd_t | |
|---|---|
| Includes board and MCU related header files. Trim command. | |
| Enumerator | |
| OPAMP_TRIM_CMD_START | Initialize trim state machine. |
| OPAMP_TRIM_CMD_NEXT_STEP | Move to next step in state machine. |
| OPAMP_TRIM_CMD_CLEAR_BIT | Clear trim bit. |

### ◆ opamp_trim_input_t

| enum opamp_trim_input_t | |
|---|---|
| Trim input. | |
| Enumerator | |
| OPAMP_TRIM_INPUT_PCH | Trim non-inverting (+) input. |
| OPAMP_TRIM_INPUT_NCH | Trim inverting (-) input. |

## 5.3.2 AI

Interfaces

### Detailed Description

AI Interfaces.

### Modules

| | |
|---|---|
| | Data Collector Interface<br>Interface for RAI Data Collector. |
| | Data Shipper Interface<br>Interface for RAI Data Shipper. |
| | Ethos-U Interface<br>Interface for Ethos-U. |

### 5.3.2.1 Data Collector Interface

Interfaces » AI

#### Detailed Description

Interface for RAI Data Collector.

## Summary

The rai data collector interface provides functionality to collect data from differnet channels using snapshot mode, data feed mode or mixed mode.

#### Data Structures

| | |
|---|---|
| struct | rai_data_collector_error_callback_args_t |
| struct | rai_data_collector_frame_buffer_t |
| struct | rai_data_collector_callback_args_t |
| struct | rai_data_collector_snapshot_cfg_t |
| struct | rai_data_collector_data_feed_cfg_t |
| struct | rai_data_collector_cfg_t |
| struct | rai_data_collector_api_t |
| struct | rai_data_collector_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | rai_data_collector_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | rai_data_collector_data_type_t |
| enum | rai_data_collector_error_event_t |

#### Data Structure Documentation

◆ **rai_data_collector_error_callback_args_t**

| |
|---|
| struct rai_data_collector_error_callback_args_t |
| Error callback function parameter |

| Data Fields | | |
|---|---|---|
| uint8_t | instance_id | Instance ID. |
| rai_data_collector_error_event_t | event | Error event. |

### ◆ rai_data_collector_frame_buffer_t

| struct rai_data_collector_frame_buffer_t | | |
|---|---|---|
| Frame buffer structure | | |
| Data Fields | | |
| void * | p_buf | Pointer to data buffer. |
| rai_data_collector_data_type_t | data_type | Data samples in the buffer. |

### ◆ rai_data_collector_callback_args_t

| struct rai_data_collector_callback_args_t | | |
|---|---|---|
| Data ready callback function parameter | | |
| Data Fields | | |
| uint8_t | frames | Number of frame buffers. |
| uint8_t | instance_id | Instance id. |
| uint32_t | frame_buf_len | Frame buffers shall have the same amount of data sample. |
| rai_data_collector_frame_buffer_t const * | p_frame_buf | Array of frame buffers. |
| void * | p_context | Pointer to the user-provided context. |

### ◆ rai_data_collector_snapshot_cfg_t

| struct rai_data_collector_snapshot_cfg_t | | |
|---|---|---|
| Snapshot mode configuration | | |
| Data Fields | | |
| uint8_t | channels | Total snapshot mode channels. |
| uint16_t | transfer_len | DTC transfer length. |
| timer_instance_t const * | p_timer | Pointer to timer instance. |
| transfer_instance_t const * | p_transfer | Pointer to DTC instance. |

### ◆ rai_data_collector_data_feed_cfg_t

| struct rai_data_collector_data_feed_cfg_t |
|---|
| Data feed mode configuration |
| Data Fields |

| | | |
|---|---|---|
| uint8_t | channels | Total data feed mode channels. |

### ◆ rai_data_collector_cfg_t

| struct rai_data_collector_cfg_t | |
|---|---|
| RAI Data Collector general configuration | |
| **Data Fields** | |
| uint32_t | channels: 8 |
| | Total number of channels. |
| | |
| uint32_t | instance_id: 8 |
| | Instance id. |
| | |
| uint32_t | virt_channels: 8 |
| | Virtual channels. |
| | |
| uint32_t | reserved: 8 |
| | Reserved. |
| | |
| uint32_t | channel_ready_mask |
| | Bitmask of configured channels. |
| | |
| uint32_t | required_frame_len |
| | Length of each frame buffer. |
| | |
| rai_data_collector_snapshot_cfg_t const * | p_snapshot_cfg |
| | Pointer to snapshot mode configuration structure. |
| | |
| rai_data_collector_data_feed_cfg_t const * | p_data_feed_cfg |
| | Pointer to data feed mode configuration structure. |

| | |
|---|---|
| | |
| void * | p_extend |
| | Pointer to extended configuration structure. |
| | |
| void(* | p_callback )(rai_data_collector_callback_args_t const *p_args) |
| | Pointer to the callback function when data is collected. |
| | |
| void(* | p_error_callback )(rai_data_collector_error_callback_args_t const *p_args) |
| | Pointer to the callback function when there is an error. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |

### ◆ rai_data_collector_api_t

| struct rai_data_collector_api_t | |
|---|---|
| RAI Data Collector interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rai_data_collector_ctrl_t *const p_ctrl, rai_data_collector_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | snapshotChannelRegister )(rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, void const *p_src) |
| | |
| fsp_err_t(* | bufferRelease )(rai_data_collector_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | bufferReset )(rai_data_collector_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | snapshotStart )(rai_data_collector_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | snapshotStop )(rai_data_collector_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | channelBufferGet )(rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, void **pp_buf) |
| | |
| fsp_err_t(* | channelWrite )(rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, const void *p_buf, uint32_t len) |
| | |
| fsp_err_t(* | close )(rai_data_collector_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rai_data_collector_api_t::open) (rai_data_collector_ctrl_t *const p_ctrl, rai_data_collector_cfg_t const *const p_cfg)

Initialize Data Collector module instance.

***Note***

*To reopen after calling this function, call rai_data_collector_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_cfg | Pointer to configuration structure |

### ◆ snapshotChannelRegister

fsp_err_t(* rai_data_collector_api_t::snapshotChannelRegister) (rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, void const *p_src)

Config transfer source address for snapshot mode channel

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_src | Pointer to transfer source address |

### ◆ bufferRelease

fsp_err_t(* rai_data_collector_api_t::bufferRelease) (rai_data_collector_ctrl_t *const p_ctrl)

Release frame buffers by upper modules

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | channel | Which snapshot mode channel |
| [in] | p_src | Chanenl source buffer address |

### ◆ bufferReset

fsp_err_t(* rai_data_collector_api_t::bufferReset) (rai_data_collector_ctrl_t *const p_ctrl)

Reset internal buffers

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ snapshotStart

fsp_err_t(* rai_data_collector_api_t::snapshotStart) (rai_data_collector_ctrl_t *const p_ctrl)

Starts snapshot mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ snapshotStop

fsp_err_t(* rai_data_collector_api_t::snapshotStop) (rai_data_collector_ctrl_t *const p_ctrl)

Stops snapshot mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ channelBufferGet

fsp_err_t(* rai_data_collector_api_t::channelBufferGet) (rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, void **pp_buf)

Get the PING or PONG buffer address for data transfer. For data feed mode only.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | channel | Which data feed mode channel |
| [out] | pp_buf | Returned buffer address |

### ◆ channelWrite

fsp_err_t(* rai_data_collector_api_t::channelWrite) (rai_data_collector_ctrl_t *const p_ctrl, uint8_t channel, const void *p_buf, uint32_t len)

Write data to frame buffer using CPU copy. For data feed mode only.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | channel | Which data feed mode channel |
| [in] | p_buf | Data buffer |
| [in] | len | Length of data buffer in data samples |

### ◆ close

fsp_err_t(* rai_data_collector_api_t::close) (rai_data_collector_ctrl_t *const p_ctrl)

Close the specified Data Collector module instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ rai_data_collector_instance_t

struct rai_data_collector_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rai_data_collector_ctrl_t * | p_ctrl | Pointer to the control structure |

| | | |
|---|---|---|
| | | for this instance. |
| rai_data_collector_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rai_data_collector_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rai_data_collector_ctrl_t

| typedef void rai_data_collector_ctrl_t |
|---|
| Data Collector control block. Allocate an instance specific control block to pass into the Data Collector API calls. |

## Enumeration Type Documentation

### ◆ rai_data_collector_data_type_t

| enum rai_data_collector_data_type_t | |
|---|---|
| Data types | |
| Enumerator | |
| RAI_DATA_COLLECTOR_DATA_TYPE_INT8_T | Signed 8-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_UINT8_T | Unsigned 8-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_INT16_T | Signed 16-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_UINT16_T | Unsigned 16-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_INT32_T | Signed 32-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_UINT32_T | Unsigned 32-bit. |
| RAI_DATA_COLLECTOR_DATA_TYPE_FLOAT | Float. |
| RAI_DATA_COLLECTOR_DATA_TYPE_DOUBLE | Double. |

### ◆ rai_data_collector_error_event_t

| enum rai_data_collector_error_event_t |
|---|
| Data Collector module error events |

### 5.3.2.2 Data Shipper Interface

Interfaces » AI

#### Detailed Description

Interface for RAI Data Shipper.

## Summary

The rai data shipper interface provides multiple communication methods.

#### Data Structures

| | |
|---|---|
| struct | rai_data_shipper_callback_args_t |
| struct | rai_data_shipper_write_params_t |
| struct | rai_data_shipper_cfg_t |
| struct | rai_data_shipper_api_t |
| struct | rai_data_shipper_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | rai_data_shipper_ctrl_t |

#### Data Structure Documentation

◆ **rai_data_shipper_callback_args_t**

| struct rai_data_shipper_callback_args_t | | |
|---|---|---|
| Callback function parameter structure | | |
| Data Fields | | |
| rm_comms_event_t | result | Whether data is sent successfully or not. |
| void * | p_context | Pointer to the user-provided context. |
| uint8_t | instance | Data collector instance ID. |

◆ **rai_data_shipper_write_params_t**

| struct rai_data_shipper_write_params_t |
|---|
| Data Shipper write function parameter structure |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | events | Events. |
| uint16_t | diagnostic_data_len | Diagnostic data length. |
| uint8_t * | p_diagnostic_data | Pointer to diagnostic data. |
| rai_data_collector_callback_args_t * | p_sensor_data | Pointer to sensor data info. |

### ◆ rai_data_shipper_cfg_t

| struct rai_data_shipper_cfg_t | |
|---|---|
| RAI Data Shipper general configuration | |
| **Data Fields** | |
| uint8_t | divider |
| | Send data on every (divider + 1) requests in case the interface bandwidth is not sufficient. |
| | |
| crc_instance_t const * | p_crc |
| | Pointer to CRC instance. |
| | |
| rm_comms_instance_t const * | p_comms |
| | Pointer to COMMS API instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_callback )(rai_data_shipper_callback_args_t *p_args) |
| | Pointer to the callback function on data sent or error. |
| | |

### ◆ rai_data_shipper_api_t

| struct rai_data_shipper_api_t | |
|---|---|
| RAI Data Shipper interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rai_data_shipper_ctrl_t *const p_ctrl, rai_data_shipper_cfg_t const *const p_cfg) |

| | |
|---|---|
| | |
| fsp_err_t(* | read )(rai_data_shipper_ctrl_t *const p_ctrl, void *const p_buf, uint32_t *const buf_len) |
| | |
| fsp_err_t(* | write )(rai_data_shipper_ctrl_t *const p_ctrl, rai_data_shipper_write_params_t const *p_write_params) |
| | |
| fsp_err_t(* | close )(rai_data_shipper_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rai_data_shipper_api_t::open) (rai_data_shipper_ctrl_t *const p_ctrl, rai_data_shipper_cfg_t const *const p_cfg)

Initialize Data Shipper module instance.

***Note***

*To reopen after calling this function, call rai_data_shipper_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_cfg | Pointer to configuration structure |

### ◆ read

fsp_err_t(* rai_data_shipper_api_t::read) (rai_data_shipper_ctrl_t *const p_ctrl, void *const p_buf, uint32_t *const buf_len)

Read data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to the location to store read data. |
| [in] | buf_len | Number of bytes to read. |

### ◆ write

| fsp_err_t(* rai_data_shipper_api_t::write) (rai_data_shipper_ctrl_t *const p_ctrl, rai_data_shipper_write_params_t const *p_write_params) |
|---|

Write data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | write_params | Pointer to write parameters structure |

### ◆ close

| fsp_err_t(* rai_data_shipper_api_t::close) (rai_data_shipper_ctrl_t *const p_ctrl) |
|---|

Close the specified Data Shipper module instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

### ◆ rai_data_shipper_instance_t

| struct rai_data_shipper_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rai_data_shipper_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rai_data_shipper_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rai_data_shipper_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rai_data_shipper_ctrl_t

| typedef void rai_data_shipper_ctrl_t |
|---|

Data Shipper control block. Allocate an instance specific control block to pass into the Data Shipper API calls.

## 5.3.2.3 Ethos-U Interface

Interfaces » AI

### Detailed Description

Interface for Ethos-U.

# Summary

The Ethos-U NPU interface provides the ability to initialize Arm Ethos-U55 NPU.

### Data Structures

| | |
|---|---|
| struct | rm_ethosu_callback_args_t |
| struct | rm_ethosu_cfg_t |
| struct | rm_ethosu_api_t |
| struct | rm_ethosu_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ethosu_ctrl_t |

### Data Structure Documentation

#### ◆ rm_ethosu_callback_args_t

| struct rm_ethosu_callback_args_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in rm_ethosu_api_t::open function in::rm_ethosu_cfg_t. |

#### ◆ rm_ethosu_cfg_t

| struct rm_ethosu_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| void(* | p_callback )(rm_ethosu_callback_args_t *p_args) |
| | Callback provided for NPU IRQ. |
| | |
| void * | p_context |
| | Placeholder for user data. |

| | |
|---|---|
| | |
| uint32_t | ipl |
| | Interrupt priority level. |
| | |
| IRQn_Type | irq |
| | NPU IRQ number. |
| | |
| bool | secure_enable |
| | Configure NPU in secure or non-secure mode. |
| | |
| bool | privilege_enable |
| | Configure NPU in privileged or non-privileged mode. |
| | |

### ◆ rm_ethosu_api_t

| struct rm_ethosu_api_t | |
|---|---|
| Ethos-U NPU interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_ethosu_ctrl_t *const p_ctrl, rm_ethosu_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | callbackSet )(rm_ethosu_ctrl_t *const p_ctrl, void(*p_callback)(rm_ethosu_callback_args_t *), void *const p_context, rm_ethosu_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(rm_ethosu_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ethosu_api_t::open) (rm_ethosu_ctrl_t *const p_ctrl, rm_ethosu_cfg_t const *const p_cfg)

Enables Ethos-U NPU module and initialize driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ callbackSet

fsp_err_t(* rm_ethosu_api_t::callbackSet) (rm_ethosu_ctrl_t *const p_ctrl, void(*p_callback)(rm_ethosu_callback_args_t *), void *const p_context, rm_ethosu_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_ethosu_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* rm_ethosu_api_t::close) (rm_ethosu_ctrl_t *const p_ctrl)

Closes the module and deinitializes Ethos-U driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_ethosu_api_t::open call. |

### ◆ rm_ethosu_instance_t

| struct rm_ethosu_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_ethosu_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ethosu_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ethosu_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ rm_ethosu_ctrl_t

| typedef void rm_ethosu_ctrl_t |
|---|
| Ethos-U NPU API control block. Allocate an instance specific control block to pass into the Ethos-U NPU API calls. |

## 5.3.3 Audio

Interfaces

### Detailed Description

Audio Interfaces.

### Modules

| | |
|---|---|
| ADPCM Decoder Interface | Interface for ADPCM decoder. |
| AUDIO PLAYBACK Interface | Interface for the Audio Playback. |
| PDM Interface | Interface for PDM audio communication. |

### 5.3.3.1 ADPCM Decoder Interface

Interfaces » Audio

#### Detailed Description

Interface for ADPCM decoder.

# Summary

The ADPCM decoder interface provides functionality to decode the 4bit ADPCM data to 16bit PCM output.

#### Data Structures

| | |
|---|---|
| struct | adpcm_decoder_cfg_t |
| struct | adpcm_decoder_api_t |
| struct | adpcm_decoder_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | adpcm_decoder_ctrl_t |

#### Data Structure Documentation

◆ **adpcm_decoder_cfg_t**

| struct adpcm_decoder_cfg_t |
|---|
| Audio Decoder general configuration |

◆ **adpcm_decoder_api_t**

| struct adpcm_decoder_api_t | |
|---|---|
| Audio Decoder interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(adpcm_decoder_ctrl_t *const p_ctrl, adpcm_decoder_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | decode )(adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes) |
| | |
| fsp_err_t(* | reset )(adpcm_decoder_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(adpcm_decoder_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* adpcm_decoder_api_t::open) (adpcm_decoder_ctrl_t *const p_ctrl, adpcm_decoder_cfg_t const *const p_cfg)

Initialize Audio Decoder device.

***Note***

*To reconfigure after calling this function, call adpcm_decoder_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_cfg | Pointer to configuration structure |

### ◆ decode

fsp_err_t(* adpcm_decoder_api_t::decode) (adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes)

Decodes the compressed data and stores it in output buffer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | p_src | Pointer to a source data buffer from which data will be picked up for decode operation. The argument must not be NULL. |
| [out] | p_dest | Pointer to the location to store the decoded data. |
| [in] | p_dest | Number of bytes to be decoded. |

### ◆ reset

fsp_err_t(* adpcm_decoder_api_t::reset) (adpcm_decoder_ctrl_t *const p_ctrl)

Resets the ADPCM driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |

◆ **close**

| fsp_err_t(* adpcm_decoder_api_t::close) (adpcm_decoder_ctrl_t *const p_ctrl) | | |
|---|---|---|
| Close the specified Audio decoder modules.<br><br>**Parameters**<br>[in] p_ctrl — Pointer to control handle structure | | |

◆ **adpcm_decoder_instance_t**

| struct adpcm_decoder_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| adpcm_decoder_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| adpcm_decoder_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| adpcm_decoder_api_t const * | p_api | Pointer to the API structure for this instance. |

**Typedef Documentation**

◆ **adpcm_decoder_ctrl_t**

| typedef void adpcm_decoder_ctrl_t |
|---|
| Audio Decoder control block. Allocate an instance specific control block to pass into the Audio Decoder API calls. |

### 5.3.3.2 AUDIO PLAYBACK Interface

Interfaces » Audio

**Detailed Description**

Interface for the Audio Playback.

Defines the API and data structures for the Audio Playback implementation.

# Summary

This module provides common interface for Audio Playback.

### Data Structures

| | |
|---|---|
| struct | audio_playback_callback_args_t |
| struct | audio_playback_cfg_t |
| struct | audio_playback_api_t |
| struct | audio_playback_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | audio_playback_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | audio_playback_event_t |

## Data Structure Documentation

### ◆ audio_playback_callback_args_t

| struct audio_playback_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| audio_playback_event_t | event | Event that triggered the callback. |

### ◆ audio_playback_cfg_t

| struct audio_playback_cfg_t | |
|---|---|
| Audio Playback configuration parameters. | |
| **Data Fields** | |
| void const * | p_extend |
| | Hardware dependent configuration. |
| | |
| void(* | p_callback )(audio_playback_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |

## Field Documentation

### ◆ p_callback

void(* audio_playback_cfg_t::p_callback) (audio_playback_callback_args_t *p_args)

Callback called when play is complete.

### ◆ p_context

void* audio_playback_cfg_t::p_context

Placeholder for user data. Passed to the user callback in audio_playback_callback_args_t.

### ◆ audio_playback_api_t

struct audio_playback_api_t

Audio Playback functions implemented by the Audio Playback drivers will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(audio_playback_ctrl_t *const p_ctrl, audio_playback_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(audio_playback_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(audio_playback_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | play )(audio_playback_ctrl_t *const p_ctrl, void const *const p_buffer, uint32_t length) |
| | |
| fsp_err_t(* | close )(audio_playback_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* audio_playback_api_t::open) (audio_playback_ctrl_t *const p_ctrl, audio_playback_cfg_t const *const p_cfg)

Open a audio playback module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to memory allocated for control block. |
| [in] | p_cfg | Pointer to the hardware configurations. |

### ◆ start

fsp_err_t(* audio_playback_api_t::start) (audio_playback_ctrl_t *const p_ctrl)

Start audio playback hardware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |

### ◆ stop

fsp_err_t(* audio_playback_api_t::stop) (audio_playback_ctrl_t *const p_ctrl)

Stop audio playback hardware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |

### ◆ play

fsp_err_t(* audio_playback_api_t::play) (audio_playback_ctrl_t *const p_ctrl, void const *const p_buffer, uint32_t length)

Play audio buffer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | p_buffer | Pointer to buffer with PCM samples to play. Data must be scaled for audio playback hardware. |
| [in] | length | Length of data in p_buffer. |

### ◆ close

fsp_err_t(* audio_playback_api_t::close) (audio_playback_ctrl_t *const p_ctrl)

Close the audio driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block initialized in audio_playback_api_t::open. |

### ◆ audio_playback_instance_t

struct audio_playback_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| audio_playback_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| audio_playback_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| audio_playback_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ audio_playback_ctrl_t

| typedef void audio_playback_ctrl_t |
|---|
| Audio Playback control block. Allocate an instance specific control block to pass into the AUDIO_PLAYBACK API calls. |

### Enumeration Type Documentation

#### ◆ audio_playback_event_t

| enum audio_playback_event_t | |
|---|---|
| Callback event types. | |
| Enumerator | |
| AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE | Audio playback complete event. |

## 5.3.3.3 PDM Interface

Interfaces » Audio

### Detailed Description

Interface for PDM audio communication.

# Summary

The pulse-density modulation(PDM) interface provides APIs and definitions for PDM audio communication.

### Data Structures

| | |
|---|---|
| struct | pdm_callback_args_t |

| | |
|---|---|
| struct | pdm_sound_detection_setting_t |
| struct | pdm_status_t |
| struct | pdm_cfg_t |
| struct | pdm_api_t |
| struct | pdm_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | pdm_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | pdm_pcm_width_t |
| enum | pdm_input_data_edge_t |
| enum | pdm_event_t |
| enum | pdm_error_t |
| enum | pdm_state_t |

## Data Structure Documentation

### ◆ pdm_callback_args_t

| struct pdm_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in pdm_api_t::open function in pdm_cfg_t. |
| pdm_event_t | event | The event can be used to identify what caused the callback (overflow or error). |
| pdm_error_t | error | The kind of error. |

### ◆ pdm_sound_detection_setting_t

| struct pdm_sound_detection_setting_t | | |
|---|---|---|
| Sound detection window setting | | |
| Data Fields | | |
| uint32_t | sound_detection_lower_limit | Sound detection lower limit (20-bit signed fixed point data) |

| | | |
|---|---|---|
| uint32_t | sound_detection_upper_limit | Sound detection upper limit (20-bit signed fixed point data) |

### ◆ pdm_status_t

| struct pdm_status_t | | |
|---|---|---|
| PDM status. | | |
| Data Fields | | |
| pdm_state_t | state | Current PDM data state. |
| bool | sound_detection_enabled | Current PDM sound detection status. |
| pdm_error_t | error | Bitfield of current PDM errors. |

### ◆ pdm_cfg_t

| struct pdm_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | unit |
| | |
| pdm_pcm_width_t | pcm_width |
| | Audio PCM data width. |
| | |
| pdm_input_data_edge_t | pcm_edge |
| | Input data select. |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | |
| void(* | p_callback )(pdm_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |
| uint8_t | dat_ipl |

| | |
|---|---|
| | Data reception interrupt priority. |
| | |
| uint8_t | sdet_ipl |
| | Sound detection interrupt priority. |
| | |
| uint8_t | err_ipl |
| | Idle/Error interrupt priority. |
| | |
| IRQn_Type | dat_irq |
| | IRQ number of data reception interrupt. |
| | |
| IRQn_Type | sdet_irq |
| | IRQ number of sound detection interrupt. |
| | |
| IRQn_Type | err_irq |
| | IRQ number of error detection interrupt. |
| | |

## Field Documentation

### ◆ unit

uint32_t pdm_cfg_t::unit

Select a unit/channel corresponding to the unit/channel number of the hardware.

### ◆ p_transfer_rx

transfer_instance_t const* pdm_cfg_t::p_transfer_rx

To use DMA for receiving link a Transfer instance here. Set to NULL if unused.

### ◆ p_callback

void(* pdm_cfg_t::p_callback) (pdm_callback_args_t *p_args)

Callback provided when an PDM ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

| void* pdm_cfg_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in pdm_callback_args_t. |

### ◆ pdm_api_t

| struct pdm_api_t | |
|---|---|
| PDM functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback) |
| | |
| fsp_err_t(* | stop )(pdm_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | soundDetectionEnable )(pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t pdm_sound_detection_setting) |
| | |
| fsp_err_t(* | soundDetectionDisable )(pdm_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | statusGet )(pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(pdm_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context, pdm_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* pdm_api_t::open) (pdm_ctrl_t *const p_ctrl, pdm_cfg_t const *const p_cfg)

Initial configuration.

**Precondition**

Peripheral clocks and any required output pins should be configured prior to calling this function.

***Note***

*To reconfigure after calling this function, call pdm_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ start

fsp_err_t(* pdm_api_t::start) (pdm_ctrl_t *const p_ctrl, void *const p_buffer, size_t const buffer_size, uint32_t const number_of_data_to_callback)

Start communication.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |
| [in] | p_buffer | Pointer to the reception buffer |
| [in] | buffer_size | Size of the buffer |
| [in] | number_of_data_to_callback | Number of the data to call the callback function from dat_isr |

### ◆ stop

fsp_err_t(* pdm_api_t::stop) (pdm_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with PDM_EVENT_IDLE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |

### ◆ soundDetectionEnable

fsp_err_t(* pdm_api_t::soundDetectionEnable) (pdm_ctrl_t *const p_ctrl, pdm_sound_detection_setting_t pdm_sound_detection_setting)

Enable sound detection with the specified settings.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |
| [in] | pdm_sound_detection_setting | Settings to configure sound detection. |

### ◆ soundDetectionDisable

fsp_err_t(* pdm_api_t::soundDetectionDisable) (pdm_ctrl_t *const p_ctrl)

Disable sound detection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |

### ◆ read

fsp_err_t(* pdm_api_t::read) (pdm_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read remaining PDM data. This function can be called during PDM_STATE_STOPPED.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |
| [in] | p_dest | Buffer to store PCM samples. Must be 4 byte aligned. |
| [in] | bytes | Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes. |

### ◆ statusGet

fsp_err_t(* pdm_api_t::statusGet) (pdm_ctrl_t *const p_ctrl, pdm_status_t *const p_status)

Get current status and store it in provided pointer p_status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |
| [out] | p_status | Current status of the driver. |

### ◆ close

fsp_err_t(* pdm_api_t::close) (pdm_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in pdm_api_t::open call for this instance. |

### ◆ callbackSet

fsp_err_t(* pdm_api_t::callbackSet) (pdm_ctrl_t *const p_ctrl, void(*p_callback)(pdm_callback_args_t *), void *const p_context, pdm_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the PDM control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ pdm_instance_t

struct pdm_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| pdm_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| pdm_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| pdm_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ pdm_ctrl_t

| typedef void pdm_ctrl_t |
|---|
| PDM control block. Allocate an instance specific control block to pass into the PDM API calls. |

## Enumeration Type Documentation

### ◆ pdm_pcm_width_t

| enum pdm_pcm_width_t | |
|---|---|
| Audio PCM width | |
| Enumerator | |
| PDM_PCM_WIDTH_20_BITS_0_18 | Using 20-bit PCM Clips [18:0] bit of the buffer and 1-bit of sign. |
| PDM_PCM_WIDTH_20_BITS_1_18 | Using 20-bit PCM Clips [18:1] bit of the buffer and 2-bits of sign. |
| PDM_PCM_WIDTH_20_BITS_2_18 | Using 20-bit PCM Clips [18:2] bit of the buffer and 3-bits of sign. |
| PDM_PCM_WIDTH_20_BITS_3_18 | Using 20-bit PCM Clips [18:3] bit of the buffer and 4-bits of sign. |
| PDM_PCM_WIDTH_16_BITS_4_18 | Using 16-bit PCM Clips [18:4] bit of the buffer and 1-bit of sign. |
| PDM_PCM_WIDTH_16_BITS_3_17 | Using 16-bit PCM Clips [17:3] bit of the buffer and 1-bit of sign. |
| PDM_PCM_WIDTH_16_BITS_2_16 | Using 16-bit PCM Clips [16:2] bit of the buffer and 1-bit of sign. |
| PDM_PCM_WIDTH_16_BITS_1_15 | Using 16-bit PCM Clips [15:1] bit of the buffer and 1-bit of sign. |
| PDM_PCM_WIDTH_16_BITS_0_14 | Using 16-bit PCM Clips [14:0] bit of the buffer and 1-bit of sign. |

### ◆ pdm_input_data_edge_t

| enum pdm_input_data_edge_t | |
|---|---|
| Input data select. | |
| Enumerator | |
| PDM_INPUT_DATA_EDGE_RISE | Rise-edge data of channel n. |
| PDM_INPUT_DATA_EDGE_FALL | Fall-edge data of channel n-1 (In case of n = 0, fall-edge data of channel 2 is selected.) |

### ◆ pdm_event_t

| enum pdm_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| PDM_EVENT_IDLE | IDLE. |
| PDM_EVENT_DATA | Data reception. |
| PDM_EVENT_SOUND_DETECTION | Sound detection. |
| PDM_EVENT_ERROR | Error. |

### ◆ pdm_error_t

| enum pdm_error_t | |
|---|---|
| Error information included in a callback function | |
| Enumerator | |
| PDM_ERROR_NONE | No error. |
| PDM_ERROR_SHORT_CIRCUIT | Short circuit detection. |
| PDM_ERROR_OVERVOLTAGE_LOWER | Overvoltage lower limit exceeded detection. |
| PDM_ERROR_OVERVOLTAGE_UPPER | Overvoltage upper limit exceeded detection. |
| PDM_ERROR_BUFFER_OVERWRITE | Buffer overwriting detection. |

### ◆ pdm_state_t

| enum pdm_state_t | |
|---|---|
| Possible status values returned by pdm_api_t::statusGet. | |
| Enumerator | |
| PDM_STATE_IN_USE | PDM is actively reading data. |
| PDM_STATE_STOPPED | PDM is stopped and not actively reading data. |

## 5.3.4 CapTouch

Interfaces

### Detailed Description

CapTouch Interfaces.

### Modules

| | |
|---|---|
| | CTSU Interface<br>Interface for Capacitive Touch Sensing Unit (CTSU) functions. |
| | Touch Middleware Interface<br>Interface for Touch Middleware functions. |

### 5.3.4.1 CTSU Interface

Interfaces » CapTouch

### Detailed Description

Interface for Capacitive Touch Sensing Unit (CTSU) functions.

# Summary

The CTSU interface provides CTSU functionality.

### Data Structures

| | |
|---|---|
| struct | ctsu_callback_args_t |
| struct | ctsu_element_cfg_t |
| struct | ctsu_auto_button_cfg_t |
| struct | ctsu_cfg_t |
| struct | ctsu_api_t |
| struct | ctsu_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ctsu_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ctsu_event_t |
| enum | ctsu_cap_t |
| enum | ctsu_txvsel_t |
| enum | ctsu_txvsel2_t |
| enum | ctsu_atune1_t |
| enum | ctsu_atune12_t |
| enum | ctsu_md_t |
| enum | ctsu_posel_t |
| enum | ctsu_ssdiv_t |
| enum | ctsu_specific_data_type_t |

## Data Structure Documentation

### ◆ ctsu_callback_args_t

| struct ctsu_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| ctsu_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. Set in ctsu_api_t::open function in ctsu_cfg_t. |

### ◆ ctsu_element_cfg_t

| struct ctsu_element_cfg_t | | |
|---|---|---|
| CTSU Configuration parameters. Element Configuration | | |
| Data Fields | | |
| ctsu_ssdiv_t | ssdiv | CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) |
| uint16_t | so | CTSU Sensor Offset Adjustment. |

| | | |
|---|---|---|
| uint8_t | snum | CTSU Measurement Count Setting. |
| uint8_t | sdpa | CTSU Base Clock Setting. |

### ◆ ctsu_auto_button_cfg_t

| struct ctsu_auto_button_cfg_t | | |
|---|---|---|
| Configuration of each automatic judgement button | | |
| Data Fields | | |
| uint8_t | elem_index | Element number used by this button for automatic judgement. |
| uint16_t | threshold | Touch/non-touch judgement threshold for automatic judgement. |
| uint16_t | hysteresis | Threshold hysteresis for chattering prevention for automatic judgement. |

### ◆ ctsu_cfg_t

| struct ctsu_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| ctsu_cap_t | cap |
| | CTSU Scan Start Trigger Select. |
| | |
| ctsu_txvsel_t | txvsel |
| | CTSU Transmission Power Supply Select. |
| | |
| ctsu_txvsel2_t | txvsel2 |
| | CTSU Transmission Power Supply Select 2 (CTSU2 Only) |
| | |
| ctsu_atune1_t | atune1 |
| | CTSU Power Supply Capacity Adjustment (CTSU Only) |
| | |
| ctsu_atune12_t | atune12 |
| | |

| | |
|---|---|
| | CTSU Power Supply Capacity Adjustment (CTSU2 Only) |
| | |
| ctsu_md_t | md |
| | CTSU Measurement Mode Select. |
| | |
| ctsu_posel_t | posel |
| | CTSU Non-Measured Channel Output Select (CTSU2 Only) |
| | |
| uint8_t | tsod |
| | TS all terminal output control for multi electrode scan. |
| | |
| uint8_t | mec_ts |
| | TS number used when using the MEC function. |
| | |
| uint8_t | mec_shield_ts |
| | TS number of active shield used when using MEC function. |
| | |
| uint8_t | tlot |
| | Number of consecutive judgements exceeding the threshold L for automatic judgement. |
| | |
| uint8_t | thot |
| | Number of consecutive judgements exceeding the threshold H for automatic judgement. |
| | |
| uint8_t | jc |
| | Judgement condition for automatic judgement. |
| | |
| uint8_t | ajmmat |

| | |
|---|---|
| | Measured value moving average number of times for automatic judgement. |
| | |
| uint8_t | ajbmat |
| | Average number of baselines for automatic judgement. |
| | |
| uint8_t | mtucfen |
| | Mutual capacity operation for automatic judgement. |
| | |
| uint8_t | ajfen |
| | Automatic judgement function enabled for automatic judgement. |
| | |
| uint8_t | autojudge_monitor_num |
| | Method number for QE monitor for automatic judgement. |
| | |
| uint8_t | majirimd |
| | JMM or VMM for automatic judgement. |
| | |
| uint8_t | ctsuchac0 |
| | TS00-TS07 enable mask. |
| | |
| uint8_t | ctsuchac1 |
| | TS08-TS15 enable mask. |
| | |
| uint8_t | ctsuchac2 |
| | TS16-TS23 enable mask. |
| | |
| uint8_t | ctsuchac3 |
| | |

| | |
|---|---|
| | TS24-TS31 enable mask. |
| | |
| uint8_t | ctsuchac4 |
| | TS32-TS39 enable mask. |
| | |
| uint8_t | ctsuchtrc0 |
| | TS00-TS07 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc1 |
| | TS08-TS15 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc2 |
| | TS16-TS23 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc3 |
| | TS24-TS31 mutual-tx mask. |
| | |
| uint8_t | ctsuchtrc4 |
| | TS32-TS39 mutual-tx mask. |
| | |
| ctsu_element_cfg_t const * | p_elements |
| | Pointer to elements configuration array. |
| | |
| uint8_t | num_rx |
| | Number of receive terminals. |
| | |
| uint8_t | num_tx |
| | Number of transmit terminals. |

| | |
|---|---|
| | |
| uint16_t | num_moving_average |
| | Number of moving average for measurement data. |
| | |
| bool | tunning_enable |
| | Initial offset tuning flag. |
| | |
| void(* | p_callback )(ctsu_callback_args_t *p_args) |
| | Callback provided when CTSUFN ISR occurs. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | DTC instance for transmit at CTSUWR. Set to NULL if unused. |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | DTC instance for receive at CTSURD. Set to NULL if unused. |
| | |
| adc_instance_t const * | p_adc_instance |
| | ADC instance for temperature correction. |
| | |
| IRQn_Type | write_irq |
| | CTSU_CTSUWR interrupt vector. |
| | |
| IRQn_Type | read_irq |
| | CTSU_CTSURD interrupt vector. |
| | |
| IRQn_Type | end_irq |
| | CTSU_CTSUFN interrupt vector. |
| | |

| | |
|---|---|
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| uint16_t | tuning_self_target_value |
| | Target self value for initial offset tuning. |
| | |
| uint16_t | tuning_mutual_target_value |
| | Target mutual value for initial offset tuning. |
| | |
| ctsu_auto_button_cfg_t const * | p_ctsu_auto_buttons |
| | Pointer to array of automatic judgement button configuration. |
| | |
| uint8_t | majority_mode |
| | Software majority mode selection. |
| | |

### ◆ ctsu_api_t

| struct ctsu_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanStart )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dataGet )(ctsu_ctrl_t *const p_ctrl, uint16_t *p_data) |
| | |
| fsp_err_t(* | scanStop )(ctsu_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | |
| fsp_err_t(* | diagnosis )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(ctsu_ctrl_t *const p_ctrl, void(*p_callback)(ctsu_callback_args_t *), void *const p_context, ctsu_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | specificDataGet )(ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type) |
| | |
| fsp_err_t(* | dataInsert )(ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data) |
| | |
| fsp_err_t(* | autoJudgementDataGet )(ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status) |
| | |
| fsp_err_t(* | offsetTuning )(ctsu_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ scanStart

fsp_err_t(* ctsu_api_t::scanStart) (ctsu_ctrl_t *const p_ctrl)

Scan start.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ dataGet

fsp_err_t(* ctsu_api_t::dataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)

Data get.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Pointer to get data array. |

### ◆ scanStop

fsp_err_t(* ctsu_api_t::scanStop) (ctsu_ctrl_t *const p_ctrl)

ScanStop.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ diagnosis

fsp_err_t(* ctsu_api_t::diagnosis) (ctsu_ctrl_t *const p_ctrl)

Diagnosis.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ callbackSet

fsp_err_t(* ctsu_api_t::callbackSet) (ctsu_ctrl_t *const p_ctrl, void(*p_callback)(ctsu_callback_args_t *), void *const p_context, ctsu_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CTSU control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ specificDataGet

fsp_err_t(* ctsu_api_t::specificDataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type)

Specific Data get.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_specific_data | Pointer to get specific data array. |
| [in] | specific_data_type | Specific data type |

### ◆ dataInsert

fsp_err_t(* ctsu_api_t::dataInsert) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data)

Data Insert.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_insert_data | Pointer to insert data. |

### ◆ autoJudgementDataGet

fsp_err_t(* ctsu_api_t::autoJudgementDataGet) (ctsu_ctrl_t *const p_ctrl, uint64_t *p_button_status)

Initialize automatic judgement and get button status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_button_status | Pointer to get button status array. |

### ◆ offsetTuning

fsp_err_t(* ctsu_api_t::offsetTuning) (ctsu_ctrl_t *const p_ctrl)

Adjust the offset value to tune the sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ ctsu_instance_t

struct ctsu_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| ctsu_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ctsu_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ctsu_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ctsu_ctrl_t

typedef void ctsu_ctrl_t

CTSU Control block. Allocate an instance specific control block to pass into the API calls.

## Enumeration Type Documentation

### ◆ ctsu_event_t

| enum ctsu_event_t | |
|---|---|
| CTSU Events for callback function | |
| Enumerator | |
| CTSU_EVENT_SCAN_COMPLETE | Normal end. |
| CTSU_EVENT_OVERFLOW | Sensor counter overflow (CTSUST.CTSUSOVF set) |
| CTSU_EVENT_ICOMP | Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set) |
| CTSU_EVENT_ICOMP1 | Abnormal sensor current (CTSUSR.ICOMP1 set) |

### ◆ ctsu_cap_t

| enum ctsu_cap_t | |
|---|---|
| CTSU Scan Start Trigger Select | |
| Enumerator | |
| CTSU_CAP_SOFTWARE | Scan start by software trigger. |
| CTSU_CAP_EXTERNAL | Scan start by external trigger. |

### ◆ ctsu_txvsel_t

| enum ctsu_txvsel_t | |
|---|---|
| CTSU Transmission Power Supply Select | |
| Enumerator | |
| CTSU_TXVSEL_VCC | VCC selected. |
| CTSU_TXVSEL_INTERNAL_POWER | Internal logic power supply selected. |

### ◆ ctsu_txvsel2_t

| enum ctsu_txvsel2_t | |
|---|---|
| CTSU Transmission Power Supply Select 2 (CTSU2 Only) | |
| Enumerator | |
| CTSU_TXVSEL_MODE | Follow TXVSEL setting. |
| CTSU_TXVSEL_VCC_PRIVATE | VCC private selected. |

### ◆ ctsu_atune1_t

| enum ctsu_atune1_t | |
|---|---|
| CTSU Power Supply Capacity Adjustment (CTSU Only) | |
| Enumerator | |
| CTSU_ATUNE1_NORMAL | Normal output (40uA) |
| CTSU_ATUNE1_HIGH | High-current output (80uA) |

### ◆ ctsu_atune12_t

| enum ctsu_atune12_t | |
|---|---|
| CTSU Power Supply Capacity Adjustment (CTSU2 Only) | |
| Enumerator | |
| CTSU_ATUNE12_80UA | High-current output (80uA) |
| CTSU_ATUNE12_40UA | Normal output (40uA) |
| CTSU_ATUNE12_20UA | Low-current output (20uA) |
| CTSU_ATUNE12_160UA | Very high-current output (160uA) |

### ◆ ctsu_md_t

| enum ctsu_md_t | |
|---|---|
| CTSU Measurement Mode Select | |
| Enumerator | |
| CTSU_MODE_SELF_MULTI_SCAN | Self-capacitance multi scan mode. |
| CTSU_MODE_MUTUAL_FULL_SCAN | Mutual capacitance full scan mode. |
| CTSU_MODE_MUTUAL_CFC_SCAN | Mutual capacitance cfc scan mode (CTSU2 Only) |
| CTSU_MODE_CURRENT_SCAN | Current scan mode (CTSU2 Only) |
| CTSU_MODE_CORRECTION_SCAN | Correction scan mode (CTSU2 Only) |
| CTSU_MODE_DIAGNOSIS_SCAN | Diagnosis scan mode. |

### ◆ ctsu_posel_t

| enum ctsu_posel_t | |
|---|---|
| CTSU Non-Measured Channel Output Select (CTSU2 Only) | |
| Enumerator | |
| CTSU_POSEL_LOW_GPIO | Output low through GPIO. |
| CTSU_POSEL_HI_Z | Hi-Z. |
| CTSU_POSEL_LOW | Setting prohibited. |
| CTSU_POSEL_SAME_PULSE | Same phase pulse output as transmission channel through the power setting by the TXVSEL[1:0] bits. |

### ◆ ctsu_ssdiv_t

| enum ctsu_ssdiv_t | |
|---|---|
| CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only) | |
| Enumerator | |
| CTSU_SSDIV_4000 | 4.00 <= Base clock frequency (MHz) |
| CTSU_SSDIV_2000 | 2.00 <= Base clock frequency (MHz) < 4.00 |
| CTSU_SSDIV_1330 | 1.33 <= Base clock frequency (MHz) < 2.00 |
| CTSU_SSDIV_1000 | 1.00 <= Base clock frequency (MHz) < 1.33 |
| CTSU_SSDIV_0800 | 0.80 <= Base clock frequency (MHz) < 1.00 |
| CTSU_SSDIV_0670 | 0.67 <= Base clock frequency (MHz) < 0.80 |
| CTSU_SSDIV_0570 | 0.57 <= Base clock frequency (MHz) < 0.67 |
| CTSU_SSDIV_0500 | 0.50 <= Base clock frequency (MHz) < 0.57 |
| CTSU_SSDIV_0440 | 0.44 <= Base clock frequency (MHz) < 0.50 |
| CTSU_SSDIV_0400 | 0.40 <= Base clock frequency (MHz) < 0.44 |
| CTSU_SSDIV_0360 | 0.36 <= Base clock frequency (MHz) < 0.40 |
| CTSU_SSDIV_0330 | 0.33 <= Base clock frequency (MHz) < 0.36 |
| CTSU_SSDIV_0310 | 0.31 <= Base clock frequency (MHz) < 0.33 |
| CTSU_SSDIV_0290 | 0.29 <= Base clock frequency (MHz) < 0.31 |
| CTSU_SSDIV_0270 | 0.27 <= Base clock frequency (MHz) < 0.29 |
| CTSU_SSDIV_0000 | 0.00 <= Base clock frequency (MHz) < 0.27 |

### ◆ ctsu_specific_data_type_t

| enum ctsu_specific_data_type_t |
|---|
| CTSU select data type for slect data get |

### 5.3.4.2 Touch Middleware Interface

Interfaces » CapTouch

#### Detailed Description

Interface for Touch Middleware functions.

# Summary

The TOUCH interface provides TOUCH functionality.

#### Data Structures

| | |
|---|---|
| struct | touch_button_cfg_t |
| struct | touch_slider_cfg_t |
| struct | touch_wheel_cfg_t |
| struct | touch_pad_cfg_t |
| struct | touch_cfg_t |
| struct | touch_sensitivity_info_t |
| struct | touch_api_t |
| struct | touch_instance_t |

#### Macros

| | |
|---|---|
| #define | TOUCH_COUNT_MAX<br>Value of Maximum count. |
| #define | TOUCH_OFF_VALUE<br>Value of Non-touch. |

#### Typedefs

| | |
|---|---|
| typedef void | touch_ctrl_t |
| typedef struct st_ctsu_callback_args | touch_callback_args_t |

#### Data Structure Documentation

### ◆ touch_button_cfg_t

| struct touch_button_cfg_t | | |
|---|---|---|
| Configuration of each button | | |
| Data Fields | | |
| uint8_t | elem_index | Element number used by this button. |
| uint16_t | threshold | Touch/non-touch judgment threshold. |
| uint16_t | hysteresis | Threshold hysteresis for chattering prevention. |

### ◆ touch_slider_cfg_t

| struct touch_slider_cfg_t | | |
|---|---|---|
| Configuration of each slider | | |
| Data Fields | | |
| uint8_t const * | p_elem_index | Element number array used by this slider. |
| uint8_t | num_elements | Number of elements used by this slider. |
| uint16_t | threshold | Position calculation start threshold value. |

### ◆ touch_wheel_cfg_t

| struct touch_wheel_cfg_t | | |
|---|---|---|
| Configuration of each wheel | | |
| Data Fields | | |
| uint8_t const * | p_elem_index | Element number array used by this wheel. |
| uint8_t | num_elements | Number of elements used by this wheel. |
| uint16_t | threshold | Position calculation start threshold value. |

### ◆ touch_pad_cfg_t

| struct touch_pad_cfg_t | | |
|---|---|---|
| Configuration of each pads | | |
| Data Fields | | |
| uint8_t const * | p_elem_index_rx | RX of element number arrays used by this pad. |
| | | |

| | | |
|---|---|---|
| uint8_t const * | p_elem_index_tx | TX of element number arrays used by this pad. |
| uint8_t | num_elements | Number of elements used by this pad. |
| uint16_t | threshold | Coordinate calculation threshold value. |
| uint16_t | rx_pixel | rx coordinate resolution |
| uint16_t | tx_pixel | tx coordinate resolution |
| uint8_t | max_touch | Maximum number of touch judgments used by the pad. |
| uint8_t | num_drift | Number of pad drift. |

### ◆ touch_cfg_t

| struct touch_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| touch_button_cfg_t const * | p_buttons | Pointer to array of button configuration. |
| touch_slider_cfg_t const * | p_sliders | Pointer to array of slider configuration. |
| touch_wheel_cfg_t const * | p_wheels | Pointer to array of wheel configuration. |
| touch_pad_cfg_t const * | p_pad | Pointer of pad configuration. |
| uint8_t | num_buttons | Number of buttons. |
| uint8_t | num_sliders | Number of sliders. |
| uint8_t | num_wheels | Number of wheels. |
| uint8_t | on_freq | The cumulative number of determinations of ON. |
| uint8_t | off_freq | The cumulative number of determinations of OFF. |
| uint16_t | drift_freq | Base value drift frequency. [0 : no use]. |
| uint16_t | cancel_freq | Maximum continuous ON. [0 : no use]. |
| uint8_t | number | Configuration number for QE monitor. |
| ctsu_instance_t const * | p_ctsu_instance | Pointer to CTSU instance. |
| uart_instance_t const * | p_uart_instance | Pointer to UART instance. |
| void * | p_context | User defined context passed into callback function. |

| | | |
|---|---|---|
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ touch_sensitivity_info_t

| struct touch_sensitivity_info_t | | |
|---|---|---|
| Configuration of each touch sensitivity information | | |
| Data Fields | | |
| uint16_t * | p_touch_sensitivity_ratio | Pointer to sensitivity ratio array. |
| uint16_t | old_threshold_ratio | Old threshold ratio. |
| uint16_t | new_threshold_ratio | New threshold ratio. |
| uint8_t | new_hysteresis_ratio | New hysteresis ratio. |

### ◆ touch_api_t

| struct touch_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | scanStart )(touch_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dataGet )(touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position) |
| | |
| fsp_err_t(* | scanStop )(ctsu_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | padDataGet )(touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch) |
| | |
| fsp_err_t(* | callbackSet )(touch_ctrl_t *const p_ctrl, void(*p_callback)(touch_callback_args_t *), void *const p_context, touch_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(touch_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | sensitivityRatioGet )(touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info) |

| | |
|---|---|
| | |
| fsp_err_t(* | thresholdAdjust )(touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info) |
| | |
| fsp_err_t(* | driftControl )(touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* touch_api_t::open) (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ scanStart

fsp_err_t(* touch_api_t::scanStart) (touch_ctrl_t *const p_ctrl)

Scan start.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ dataGet

fsp_err_t(* touch_api_t::dataGet) (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)

Data get.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_button_status | Pointer to get data bitmap. |
| [out] | p_slider_position | Pointer to get data array. |
| [out] | p_wheel_position | Pointer to get data array. |

### ◆ scanStop

fsp_err_t(* touch_api_t::scanStop) (ctsu_ctrl_t *const p_ctrl)

ScanStop.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ padDataGet

fsp_err_t(* touch_api_t::padDataGet) (touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch)

pad data get.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_pad_rx_coordinate | Pointer to get coordinate of receiver side. |
| [out] | p_pad_tx_coordinate | Pointer to get coordinate of transmitter side. |
| [out] | p_pad_num_touch | Pointer to get touch count. |

### ◆ callbackSet

fsp_err_t(* touch_api_t::callbackSet) (touch_ctrl_t *const p_ctrl, void(*p_callback)(touch_callback_args_t *), void *const p_context, touch_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CTSU control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* touch_api_t::close) (touch_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ sensitivityRatioGet

fsp_err_t(* touch_api_t::sensitivityRatioGet) (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)

Sensitivity ratio get.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in,out] | p_touch_sensitivity_info | Pointer to touch sensitivity structure. |

### ◆ thresholdAdjust

fsp_err_t(* touch_api_t::thresholdAdjust) (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)

Threshold adjust.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_touch_sensitivity_info | Pointer to touch sensitivity structure. |

### ◆ driftControl

fsp_err_t(* touch_api_t::driftControl) (touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq)

Drift control.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | input_drift_freq | Drift frequency value. |

### ◆ touch_instance_t

struct touch_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| touch_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| touch_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| touch_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ touch_ctrl_t

| typedef void touch_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

#### ◆ touch_callback_args_t

| typedef struct st_ctsu_callback_args touch_callback_args_t |
|---|
| Callback function parameter data |

## 5.3.5 Connectivity

Interfaces

### Detailed Description

Connectivity Interfaces.

### Modules

| | |
|---|---|
| | CAN Interface<br>Interface for CAN peripheral. |
| | CEC Interface<br>Interface for CEC peripheral. |
| | Communications Middleware Interface<br>Interface for Communications Middleware functions. |

| | |
|---|---|
| | I2C Master Interface<br>Interface for I2C master communication. |
| | I2C Slave Interface<br>Interface for I2C slave communication. |
| | I2S Interface<br>Interface for I2S audio communication. |
| | I3C Interface<br>Interface for I3C. |
| | LIN Interface<br>Interface for LIN communications. |
| | SMCI Interface<br>Interface for SMCI communications. |
| | SPI Interface<br>Interface for SPI communications. |
| | UART Interface<br>Interface for UART communications. |
| | USB HAUD Interface<br>Interface for USB HAUD functions. |
| | USB HCDC Interface<br>Interface for USB HCDC functions. |
| | USB HHID Interface<br>Interface for USB HHID functions. |
| | USB HMSC Interface |

| | |
|---|---|
| | Interface for USB HMSC functions. |
| | USB Interface<br>Interface for USB functions. |
| | USB PAUD Interface<br>Interface for USB PAUD functions. |
| | USB PCDC Interface<br>Interface for USB PCDC functions. |
| | USB PHID Interface<br>Interface for USB PHID functions. |
| | USB PMSC Interface<br>Interface for USB PMSC functions. |
| | USB PPRN Interface<br>Interface for USB PPRN functions. |
| | USB TYPEC Interface<br>Interface for USB Type-C functions. |

### 5.3.5.1 CAN Interface

Interfaces » Connectivity

#### Detailed Description

Interface for CAN peripheral.

## Summary

The CAN interface provides common APIs for CAN HAL drivers. CAN interface supports following features.

- Full-duplex CAN communication
- Generic CAN parameter setting
- Interrupt driven transmit/receive processing
- Callback function support with returning event code
- Hardware resource locking during a transaction

### Data Structures

| | |
|---|---|
| struct | can_info_t |
| struct | can_bit_timing_cfg_t |
| struct | can_frame_t |
| struct | can_callback_args_t |
| struct | can_cfg_t |
| struct | can_api_t |
| struct | can_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | can_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | can_event_t |
| enum | can_operation_mode_t |
| enum | can_test_mode_t |
| enum | can_id_mode_t |
| enum | can_frame_type_t |

## Data Structure Documentation

### ◆ can_info_t

| struct can_info_t | | |
|---|---|---|
| CAN status info | | |
| Data Fields | | |
| uint32_t | status | Useful information from the CAN status register. |
| uint32_t | rx_mb_status | RX Message Buffer New Data flags. |

| | | |
|---|---|---|
| uint32_t | rx_fifo_status | RX FIFO Empty flags. |
| uint8_t | error_count_transmit | Transmit error count. |
| uint8_t | error_count_receive | Receive error count. |
| uint32_t | error_code | Error code, cleared after reading. |

### ◆ can_bit_timing_cfg_t

| struct can_bit_timing_cfg_t | | |
|---|---|---|
| CAN bit rate configuration. | | |
| Data Fields | | |
| uint32_t | baud_rate_prescaler | Baud rate prescaler. Valid values: 1 - 1024. |
| uint32_t | time_segment_1 | Time segment 1 control. |
| uint32_t | time_segment_2 | Time segment 2 control. |
| uint32_t | synchronization_jump_width | Synchronization jump width. |

### ◆ can_frame_t

| struct can_frame_t | | |
|---|---|---|
| CAN data Frame | | |
| Data Fields | | |
| uint32_t | id | CAN ID. |
| can_id_mode_t | id_mode | Standard or Extended ID (IDE). |
| can_frame_type_t | type | Frame type (RTR). |
| uint8_t | data_length_code | CAN Data Length Code (DLC). |
| uint32_t | options | Implementation-specific options. |
| uint8_t | data[CAN_DATA_BUFFER_LENGTH] | CAN data. |

### ◆ can_callback_args_t

| struct can_callback_args_t | | |
|---|---|---|
| CAN callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| can_event_t | event | Event code. |
| uint32_t | error | Error code. |
| union can_callback_args_t | __unnamed__ | |
| | | |

| | | |
|---|---|---|
| void * | p_context | Context provided to user during callback. |
| can_frame_t | frame | Received frame data. |

### ◆ can_cfg_t

| struct can_cfg_t | |
|---|---|
| CAN Configuration | |
| **Data Fields** | |
| uint32_t | channel |
| | CAN channel. |
| | |
| can_bit_timing_cfg_t * | p_bit_timing |
| | CAN bit timing. |
| | |
| void(* | p_callback )(can_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined callback context. |
| | |
| void const * | p_extend |
| | CAN hardware dependent configuration. |
| | |
| uint8_t | ipl |
| | Error/Transmit/Receive interrupt priority. |
| | |
| IRQn_Type | error_irq |
| | Error IRQ number. |
| | |
| IRQn_Type | rx_irq |
| | Receive IRQ number. |

| | |
|---|---|
| | |
| IRQn_Type | tx_irq |
| | Transmit IRQ number. |
| | |

### ◆ can_api_t

| struct can_api_t | |
|---|---|
| Shared Interface definition for CAN | |
| **Data Fields** | |
| fsp_err_t(* | open )(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame) |
| | |
| fsp_err_t(* | read )(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame) |
| | |
| fsp_err_t(* | close )(can_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | modeTransition )(can_ctrl_t *const p_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode) |
| | |
| fsp_err_t(* | infoGet )(can_ctrl_t *const p_ctrl, can_info_t *const p_info) |
| | |
| fsp_err_t(* | callbackSet )(can_ctrl_t *const p_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

Open function for CAN device

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the CAN control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to CAN configuration structure. All elements of this structure must be set by user. |

### ◆ write

fsp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Write function for CAN device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |
| [in] | buffer | Buffer number (mailbox or message buffer) to write to. |
| [in] | p_frame | Pointer for frame of CAN ID, DLC, data and frame type to write. |

### ◆ read

fsp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Read function for CAN device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |
| [in] | buffer | Message buffer (number) to read from. |
| [in] | p_frame | Pointer to store the CAN ID, DLC, data and frame type. |

### ◆ close

fsp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)

Close function for CAN device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |

### ◆ modeTransition

fsp_err_t(* can_api_t::modeTransition) (can_ctrl_t *const p_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)

Mode Transition function for CAN device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAN control block. |
| [in] | operation_mode | Destination CAN operation state. |
| [in] | test_mode | Destination CAN test state. |

### ◆ infoGet

fsp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)

Get CAN channel info.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Handle for channel (pointer to channel control block) |
| [out] | p_info | Memory address to return channel specific data to. |

### ◆ callbackSet

| fsp_err_t(* can_api_t::callbackSet) (can_ctrl_t *const p_ctrl, void(*p_callback)(can_callback_args_t *), void *const p_context, can_callback_args_t *const p_callback_memory) |
|---|

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in can_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ can_instance_t

| struct can_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| can_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| can_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| can_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ can_ctrl_t

| typedef void can_ctrl_t |
|---|
| CAN control block. Allocate an instance specific control block to pass into the CAN API calls. |

## Enumeration Type Documentation

### ◆ can_event_t

| enum can_event_t | |
|---|---|
| CAN event codes | |
| Enumerator | |
| CAN_EVENT_ERR_WARNING | Error Warning event. |
| CAN_EVENT_ERR_PASSIVE | Error Passive event. |
| CAN_EVENT_ERR_BUS_OFF | Bus Off event. |
| CAN_EVENT_BUS_RECOVERY | Bus Off Recovery event. |
| CAN_EVENT_MAILBOX_MESSAGE_LOST | Mailbox has been overrun. |
| CAN_EVENT_ERR_BUS_LOCK | Bus lock detected (32 consecutive dominant bits). |
| CAN_EVENT_ERR_CHANNEL | Channel error has occurred. |
| CAN_EVENT_TX_ABORTED | Transmit abort event. |
| CAN_EVENT_RX_COMPLETE | Receive complete event. |
| CAN_EVENT_TX_COMPLETE | Transmit complete event. |
| CAN_EVENT_ERR_GLOBAL | Global error has occurred. |
| CAN_EVENT_TX_FIFO_EMPTY | Transmit FIFO is empty. |
| CAN_EVENT_FIFO_MESSAGE_LOST | Receive FIFO overrun. |

### ◆ can_operation_mode_t

| enum can_operation_mode_t | |
|---|---|
| CAN Operation modes | |
| Enumerator | |
| CAN_OPERATION_MODE_NORMAL | CAN Normal Operation Mode. |
| CAN_OPERATION_MODE_RESET | CAN Reset Operation Mode. |
| CAN_OPERATION_MODE_HALT | CAN Halt Operation Mode. |
| CAN_OPERATION_MODE_SLEEP | CAN Sleep Operation Mode. |

### ◆ can_test_mode_t

| enum can_test_mode_t | |
|---|---|
| CAN Test modes | |
| Enumerator | |
| CAN_TEST_MODE_DISABLED | CAN Test Mode Disabled. |
| CAN_TEST_MODE_LISTEN | CAN Test Listen Mode. |
| CAN_TEST_MODE_LOOPBACK_EXTERNAL | CAN Test External Loopback Mode. |
| CAN_TEST_MODE_LOOPBACK_INTERNAL | CAN Test Internal Loopback Mode. |
| CAN_TEST_MODE_INTERNAL_BUS | CANFD Internal CAN Bus Communication Test Mode. |

### ◆ can_id_mode_t

| enum can_id_mode_t | |
|---|---|
| CAN ID modes | |
| Enumerator | |
| CAN_ID_MODE_STANDARD | Standard IDs of 11 bits used. |
| CAN_ID_MODE_EXTENDED | Extended IDs of 29 bits used. |

### ◆ can_frame_type_t

| enum can_frame_type_t | |
|---|---|
| CAN frame types | |
| Enumerator | |
| CAN_FRAME_TYPE_DATA | Data frame. |
| CAN_FRAME_TYPE_REMOTE | Remote frame. |

### 5.3.5.2 CEC Interface

Interfaces » Connectivity

#### Detailed Description

Interface for CEC peripheral.

# Summary

The CEC interface provides common APIs for CEC HAL drivers and supports the following features:

- Opening and closing the CEC module.
- Allocation for full range of local address settings (TV, Recording Device, Playback Device, etc.)
- Supports a user-callback function (required), invoked when transmit, receive, or error interrupts are received.

#### Data Structures

| | |
|---|---|
| union | cec_message_t |
| struct | cec_callback_args_t |
| struct | cec_cfg_t |
| struct | cec_api_t |
| struct | cec_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | cec_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | cec_addr_t |
| enum | cec_clock_source_t |
| enum | cec_state_t |
| enum | cec_error_t |
| enum | cec_event_t |

## Data Structure Documentation

### ◆ cec_message_t

| union cec_message_t | | |
|---|---|---|
| CEC message | | |
| Data Fields | | |
| struct cec_message_t | __unnamed__ | |
| uint8_t | raw_data[CEC_DATA_BUFFER_LENGTH+2 *sizeof(uint8_t)] | Contiguous raw data. |

### ◆ cec_callback_args_t

| struct cec_callback_args_t | | |
|---|---|---|
| CEC callback parameter definition | | |
| Data Fields | | |
| cec_event_t | event | Event code. |
| void * | p_context | Context provided to user during callback. |
| bool | addr_match | Local addresss matches message destination. |
| uint8_t | data_byte | Received data byte (INTDA) |
| cec_status_t | status | CEC Module status data. |
| cec_error_t | errors | Error code bitfield. |

### ◆ cec_cfg_t

| struct cec_cfg_t | |
|---|---|
| CEC Configuration | |
| **Data Fields** | |
| cec_timing_t const * | bit_timing_cfg |
| | CEC Bit Timing Configuration. |

| | |
|---|---|
| | |
| uint16_t | rx_data_sample_time |
| | Receive Data Sample Time Setting. |
| | |
| uint16_t | rx_data_bit_reference_width |
| | Receive Data Bit Reference Width. |
| | |
| void(* | p_callback )(cec_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined callback context. |
| | |
| uint8_t | ipl |
| | Error/Data/Message interrupt priority level. |
| | |
| IRQn_Type | error_irq |
| | Error IRQ number. |
| | |
| IRQn_Type | data_irq |
| | Data IRQ number. |
| | |
| IRQn_Type | msg_irq |
| | Communication Complete IRQ number. |
| | |
| void * | p_extend |
| | Pointer to extended configuration structure. |
| | |

### ◆ cec_api_t

| struct cec_api_t | |
|---|---|
| Shared Interface definition for CEC | |
| **Data Fields** | |
| fsp_err_t(* | open )(cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(cec_ctrl_t *const p_ctrl, cec_addr_t local_address) |
| | |
| fsp_err_t(* | write )(cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size) |
| | |
| fsp_err_t(* | close )(cec_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(cec_ctrl_t *const p_ctrl, cec_status_t *const p_status) |
| | |
| fsp_err_t(* | callbackSet )(cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void *const p_context, cec_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* cec_api_t::open) (cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg)

Open function for CEC device

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the CEC control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to CEC configuration structure. All elements of this structure must be set by user. |

### ◆ mediaInit

fsp_err_t(* cec_api_t::mediaInit) (cec_ctrl_t *const p_ctrl, cec_addr_t local_address)

Initializes the CEC device. May be called any time after the CEC module has been opened. This API blocks until the device initialization procedure is complete.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CEC instance control block. |
| [out] | local_address | Desired Logical address for local device. |

### ◆ write

fsp_err_t(* cec_api_t::write) (cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size)

Write function for CEC device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CEC instance control block |
| [in] | p_message | Message data |
| [in] | message_size | Total size of entire message |

### ◆ close

fsp_err_t(* cec_api_t::close) (cec_ctrl_t *const p_ctrl)

Close function for CEC device

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CEC instance control block |
| [out] | p_message | Message data |

### ◆ statusGet

fsp_err_t(* cec_api_t::statusGet) (cec_ctrl_t *const p_ctrl, cec_status_t *const p_status)

Get CEC channel info.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CEC instance control block |
| [out] | p_status | Memory address to return channel specific data to. |

### ◆ callbackSet

fsp_err_t(* cec_api_t::callbackSet) (cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void *const p_context, cec_callback_args_t *const p_callback_memory)

Specify callback function, optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in cec_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure cec be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ cec_instance_t

struct cec_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| cec_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| cec_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| cec_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ cec_ctrl_t

| typedef void cec_ctrl_t |
| --- |
| CEC control block. Allocate an instance specific control block to pass into the CEC API calls. |

## Enumeration Type Documentation

### ◆ cec_addr_t

| enum cec_addr_t | |
| --- | --- |
| CEC Addresses | |
| Enumerator | |
| CEC_ADDR_TV | CEC Address for TV. |
| CEC_ADDR_RECORDING_DEVICE_1 | CEC Address for Recording Device 1. |
| CEC_ADDR_RECORDING_DEVICE_2 | CEC Address for Recording Device 2. |
| CEC_ADDR_TUNER_1 | CEC Address for Tuner 1. |
| CEC_ADDR_PLAYBACK_DEVICE_1 | CEC Address for Playback Device 1. |
| CEC_ADDR_AUDIO_SYSTEM | CEC Address for Audio System. |
| CEC_ADDR_TUNER_2 | CEC Address for Tuner 2. |
| CEC_ADDR_TUNER_3 | CEC Address for Tuner 3. |
| CEC_ADDR_PLAYBACK_DEVICE_2 | CEC Address for Playback Device 2. |
| CEC_ADDR_RECORDING_DEVICE_3 | CEC Address for Recording Device 3. |
| CEC_ADDR_TUNER_4 | CEC Address for Tuner 4. |
| CEC_ADDR_PLAYBACK_DEVICE_3 | CEC Address for Playback Device 3. |
| CEC_ADDR_SPECIFIC_USE | CEC Address for Specific Use. |
| CEC_ADDR_UNREGISTERED | CEC Address for Unregistered Devices. |
| CEC_ADDR_BROADCAST | CEC Broadcast message. |

### ◆ cec_clock_source_t

| enum cec_clock_source_t | |
|---|---|
| CEC Source Clock | |
| Enumerator | |
| CEC_CLOCK_SOURCE_PCLKB_DIV_32 | PCLKB / 32 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_PCLKB_DIV_64 | PCLKB / 64 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_PCLKB_DIV_128 | PCLKB / 128 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_PCLKB_DIV_256 | PCLKB / 256 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_PCLKB_DIV_512 | PCLKB / 512 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_PCLKB_DIV_1024 | PCLKB / 1024 is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_CECCLK | CECCLK is the source of the CEC Clock. |
| CEC_CLOCK_SOURCE_CECCLK_DIV_256 | CECCLK / 256 is the source of the CEC Clock. |

### ◆ cec_state_t

| enum cec_state_t | |
|---|---|
| CEC State | |
| Enumerator | |
| CEC_STATE_UNINIT | Module requires initialization. |
| CEC_STATE_READY | Module ready for operation. |
| CEC_STATE_TX_ACTIVE | Transmit in progress, either direct or broadcast. |
| CEC_STATE_RX_ACTIVE | Receive in progress, either direct or broadcast. |
| CEC_STATE_BUSY | CEC Signal Free Time has not yet elapsed. |

### ◆ cec_error_t

| enum cec_error_t | |
|---|---|
| CEC Error Code | |
| Enumerator | |
| CEC_ERROR_NONE | No errors currently active. |
| CEC_ERROR_OERR | Overrun error. |
| CEC_ERROR_UERR | Unterrun Error. |
| CEC_ERROR_ACKERR | ACK Error. |
| CEC_ERROR_TERR | Timing Error. |
| CEC_ERROR_TXERR | Transmission Error. |
| CEC_ERROR_AERR | Bus arbitration Loss. |
| CEC_ERROR_BLERR | Bus lock error. |
| CEC_ERROR_ADDR | Address allocation error. |

### ◆ cec_event_t

| enum cec_event_t | |
|---|---|
| CEC event codes | |
| Enumerator | |
| CEC_EVENT_RX_DATA | Receive Data byte event. |
| CEC_EVENT_RX_COMPLETE | Receive complete event. |
| CEC_EVENT_TX_COMPLETE | Transmit complete event. |
| CEC_EVENT_READY | CEC Address allocated and module is now ready. |
| CEC_EVENT_ERR | Error has occurred. |

## 5.3.5.3 Communications Middleware Interface

Interfaces » Connectivity

### Detailed Description

Interface for Communications Middleware functions.

# Summary

The Communications interface provides multiple communications functionality.

### Data Structures

| | |
|---|---|
| struct | rm_comms_write_read_params_t |
| struct | rm_comms_callback_args_t |
| struct | rm_comms_cfg_t |
| struct | rm_comms_api_t |
| struct | rm_comms_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_comms_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_comms_event_t |

## Data Structure Documentation

### ◆ rm_comms_write_read_params_t

| |
|---|
| struct rm_comms_write_read_params_t |
| Struct to pack params for writeRead |

### ◆ rm_comms_callback_args_t

| |
|---|
| struct rm_comms_callback_args_t |
| Communications middleware callback parameter definition |

### ◆ rm_comms_cfg_t

| |
|---|
| struct rm_comms_cfg_t |
| Communications middleware configuration block |
| **Data Fields** |

| | |
|---|---|
| uint32_t | semaphore_timeout |
| | Timeout for read/write. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void const * | p_lower_level_cfg |
| | Pointer to lower level driver configuration structure. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void(* | p_callback )(rm_comms_callback_args_t *p_args) |
| | Pointer to callback function, mostly used if using non-blocking functionality. |
| | |

### ◆ rm_comms_api_t

| struct rm_comms_api_t | |
|---|---|
| COMM APIs | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_comms_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | writeRead )(rm_comms_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_comms_write_read_params_t write_read_params) |
| | |
| fsp_err_t(* | callbackSet )(rm_comms_ctrl_t *const p_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_comms_api_t::open) (rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_comms_api_t::close) (rm_comms_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* rm_comms_api_t::read) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_dest | Pointer to the location to store read data. |
| [in] | bytes | Number of bytes to read. |

### ◆ write

fsp_err_t(* rm_comms_api_t::write) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Write data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_src | Pointer to the location to get write data from. |
| [in] | bytes | Number of bytes to write. |

### ◆ writeRead

fsp_err_t(* rm_comms_api_t::writeRead) (rm_comms_ctrl_t *const p_ctrl, rm_comms_write_read_params_t write_read_params)

Write bytes over comms followed by a read, will have a struct for params.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | write_read_params | Parameters structure. |

### ◆ callbackSet

fsp_err_t(* rm_comms_api_t::callbackSet) (rm_comms_ctrl_t *const p_ctrl, void(*p_callback)(rm_comms_callback_args_t *), void *const p_context)

Specify callback function and optional context pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |

### ◆ rm_comms_instance_t

struct rm_comms_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

## Typedef Documentation

### ◆ rm_comms_ctrl_t

| typedef void rm_comms_ctrl_t |
| --- |
| Communications control block. Allocate an instance specific control block to pass into the Communications API calls. |

#### Enumeration Type Documentation

### ◆ rm_comms_event_t

| enum rm_comms_event_t |
| --- |
| Event in the callback function |

### 5.3.5.4 I2C Master Interface

Interfaces » Connectivity

#### Detailed Description

Interface for I2C master communication.

# Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which can return an event code

#### Data Structures

| | |
| --- | --- |
| struct | i2c_master_callback_args_t |
| struct | i2c_master_status_t |
| struct | i2c_master_cfg_t |
| struct | i2c_master_api_t |
| struct | i2c_master_instance_t |

#### Typedefs

| | |
| --- | --- |
| typedef void | i2c_master_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | i2c_master_rate_t |
| enum | i2c_master_addr_mode_t |
| enum | i2c_master_event_t |

## Data Structure Documentation

### ◆ i2c_master_callback_args_t

| struct i2c_master_callback_args_t | | |
|---|---|---|
| I2C callback parameter definition | | |
| Data Fields | | |
| void * | p_context | Pointer to user-provided context. |
| i2c_master_event_t | event | Event code. |

### ◆ i2c_master_status_t

| struct i2c_master_status_t | | |
|---|---|---|
| I2C status indicators | | |
| Data Fields | | |
| bool | open | True if driver is open. |

### ◆ i2c_master_cfg_t

| struct i2c_master_cfg_t | |
|---|---|
| I2C configuration block | |
| **Data Fields** | |
| uint8_t | channel |
| | Identifier recognizable by implementation. More... |
| | |
| i2c_master_rate_t | rate |
| | Device's maximum clock rate from enum i2c_rate_t. |
| | |
| uint32_t | slave |
| | The address of the slave device. |
| | |
| i2c_master_addr_mode_t | addr_mode |

| | |
|---|---|
| | Indicates how slave fields should be interpreted. |
| | |
| uint8_t | ipl |
| | Interrupt priority level. Same for RXI, TXI, TEI and ERI. |
| | |
| IRQn_Type | rxi_irq |
| | Receive IRQ number. |
| | |
| IRQn_Type | txi_irq |
| | Transmit IRQ number. |
| | |
| IRQn_Type | tei_irq |
| | Transmit end IRQ number. |
| | |
| IRQn_Type | eri_irq |
| | Error IRQ number. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | Transfer instance for I2C transmit. Set to NULL if unused. More... |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | Transfer instance for I2C receive. Set to NULL if unused. |
| | |
| void(* | p_callback )(i2c_master_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |

| | |
|---|---|
| void const * | p_extend |
| | Any configuration data needed by the hardware. More... |

## Field Documentation

### ◆ channel

uint8_t i2c_master_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

### ◆ p_transfer_tx

transfer_instance_t const* i2c_master_cfg_t::p_transfer_tx

Transfer instance for I2C transmit. Set to NULL if unused.

Transfer API support

### ◆ p_callback

void(* i2c_master_cfg_t::p_callback) (i2c_master_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

### ◆ p_extend

void const* i2c_master_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

### ◆ i2c_master_api_t

struct i2c_master_api_t

Interface definition for I2C access as master

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart) |
| | |
| fsp_err_t(* | write )(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t |

| | |
|---|---|
| | const bytes, bool const restart) |
| | |
| fsp_err_t(* | abort )(i2c_master_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | slaveAddressSet )(i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode) |
| | |
| fsp_err_t(* | callbackSet )(i2c_master_ctrl_t *const p_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | statusGet )(i2c_master_ctrl_t *const p_ctrl, i2c_master_status_t *p_status) |
| | |
| fsp_err_t(* | close )(i2c_master_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* i2c_master_api_t::open) (i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

Opens the I2C Master driver and initializes the hardware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements are set here. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ read

fsp_err_t(* i2c_master_api_t::read) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C Master device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_master_api_t::open call. |
| [in] | p_dest | Pointer to the location to store read data. |
| [in] | bytes | Number of bytes to read. |
| [in] | restart | Specify if the restart condition should be issued after reading. |

### ◆ write

fsp_err_t(* i2c_master_api_t::write) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

Performs a write operation on an I2C Master device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_master_api_t::open call. |
| [in] | p_src | Pointer to the location to get write data from. |
| [in] | bytes | Number of bytes to write. |
| [in] | restart | Specify if the restart condition should be issued after writing. |

### ◆ abort

fsp_err_t(* i2c_master_api_t::abort) (i2c_master_ctrl_t *const p_ctrl)

Performs a reset of the peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_master_api_t::open call. |

### ◆ slaveAddressSet

fsp_err_t(* i2c_master_api_t::slaveAddressSet) (i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)

Sets address of the slave device without reconfiguring the bus.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_master_api_t::open call. |
| [in] | slave_address | Address of the slave device. |
| [in] | address_mode | Addressing mode. |

### ◆ callbackSet

fsp_err_t(* i2c_master_api_t::callbackSet) (i2c_master_ctrl_t *const p_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void *const p_context, i2c_master_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IIC Master control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ statusGet

fsp_err_t(* i2c_master_api_t::statusGet) (i2c_master_ctrl_t *const p_ctrl, i2c_master_status_t *p_status)

Gets the status of the configured I2C device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IIC Master control block. |
| [out] | p_status | Pointer to store current status. |

### ◆ close

| fsp_err_t(* i2c_master_api_t::close) (i2c_master_ctrl_t *const p_ctrl) |
|---|

Closes the driver and releases the I2C Master device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_master_api_t::open call. |

### ◆ i2c_master_instance_t

| struct i2c_master_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| i2c_master_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| i2c_master_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i2c_master_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ i2c_master_ctrl_t

| typedef void i2c_master_ctrl_t |
|---|

I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

## Enumeration Type Documentation

### ◆ i2c_master_rate_t

| enum i2c_master_rate_t | |
|---|---|
| Communication speed options | |
| Enumerator | |
| I2C_MASTER_RATE_STANDARD | 100 kHz |
| I2C_MASTER_RATE_FAST | 400 kHz |
| I2C_MASTER_RATE_FASTPLUS | 1 MHz |
| I2C_MASTER_RATE_HIGHSPEED | 3.4 MHz |

### ◆ i2c_master_addr_mode_t

| enum i2c_master_addr_mode_t | |
|---|---|
| Addressing mode options | |
| Enumerator | |
| I2C_MASTER_ADDR_MODE_7BIT | Use 7-bit addressing mode. |
| I2C_MASTER_ADDR_MODE_10BIT | Use 10-bit addressing mode. |

### ◆ i2c_master_event_t

| enum i2c_master_event_t | |
|---|---|
| Callback events | |
| Enumerator | |
| I2C_MASTER_EVENT_ABORTED | A transfer was aborted. |
| I2C_MASTER_EVENT_RX_COMPLETE | A receive operation was completed successfully. |
| I2C_MASTER_EVENT_TX_COMPLETE | A transmit operation was completed successfully. |
| I2C_MASTER_EVENT_START | I2C sent a start condition. |
| I2C_MASTER_EVENT_BYTE_ACK | I2C finished sending/receiving 1 data byte. |

### 5.3.5.5 I2C Slave Interface

Interfaces » Connectivity

#### Detailed Description

Interface for I2C slave communication.

# Summary

The I2C slave interface provides a common API for I2C HAL drivers. The I2C slave interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which returns a event codes

#### Data Structures

| | |
|---|---|
| struct | i2c_slave_callback_args_t |
| struct | i2c_slave_cfg_t |
| struct | i2c_slave_api_t |
| struct | i2c_slave_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | i2c_slave_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | i2c_slave_rate_t |
| enum | i2c_slave_addr_mode_t |
| enum | i2c_slave_event_t |

#### Data Structure Documentation

◆ **i2c_slave_callback_args_t**

| struct i2c_slave_callback_args_t | | |
|---|---|---|
| I2C callback parameter definition | | |
| Data Fields | | |
| void * | p_context | Pointer to user-provided context. |
| uint32_t | bytes | Number of received/transmitted bytes in buffer. |

| | | |
|---|---|---|
| i2c_slave_event_t | event | Event code. |

### ◆ i2c_slave_cfg_t

| struct i2c_slave_cfg_t | |
|---|---|
| I2C configuration block | |
| **Data Fields** | |
| uint8_t | channel |
| | Identifier recognizable by implementation. More... |
| | |
| i2c_slave_rate_t | rate |
| | Device's maximum clock rate from enum i2c_rate_t. |
| | |
| uint16_t | slave |
| | The address of the slave device. |
| | |
| i2c_slave_addr_mode_t | addr_mode |
| | Indicates how slave fields should be interpreted. |
| | |
| bool | general_call_enable |
| | Allow a General call from master. |
| | |
| IRQn_Type | rxi_irq |
| | Receive IRQ number. |
| | |
| IRQn_Type | txi_irq |
| | Transmit IRQ number. |
| | |
| IRQn_Type | tei_irq |
| | Transmit end IRQ number. |
| | |

| | |
|---|---|
| IRQn_Type | eri_irq |
| | Error IRQ number. |
| | |
| uint8_t | ipl |
| | Interrupt priority level for receive, transmit, and transmit end interrupts. |
| | |
| uint8_t | eri_ipl |
| | Interrupt priority level for error interrupt. |
| | |
| bool | clock_stretching_enable |
| | Low Hold SCL during reception for the period between the 9th and the 1st clock cycle. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | Transfer instance for I2C transmit. Set to NULL if unused. More... |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | Transfer instance for I2C receive. Set to NULL if unused. |
| | |
| void(* | p_callback )(i2c_slave_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Any configuration data needed by the hardware. More... |

## Field Documentation

### ◆ channel

uint8_t i2c_slave_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

### ◆ p_transfer_tx

transfer_instance_t const* i2c_slave_cfg_t::p_transfer_tx

Transfer instance for I2C transmit. Set to NULL if unused.

Transfer API support

### ◆ p_callback

void(* i2c_slave_cfg_t::p_callback) (i2c_slave_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

### ◆ p_extend

void const* i2c_slave_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

### ◆ i2c_slave_api_t

struct i2c_slave_api_t

Interface definition for I2C access as slave

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | callbackSet )(i2c_slave_ctrl_t *const p_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, |

| | |
|---|---|
| | i2c_slave_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(i2c_slave_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* i2c_slave_api_t::open) (i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)

Opens the I2C Slave driver and initializes the hardware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements are set here. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ read

fsp_err_t(* i2c_slave_api_t::read) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Performs a read operation on an I2C Slave device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |
| [in] | p_dest | Pointer to the location to store read data. |
| [in] | bytes | Number of bytes to read. |

### ◆ write

fsp_err_t(* i2c_slave_api_t::write) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Performs a write operation on an I2C Slave device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |
| [in] | p_src | Pointer to the location to get write data from. |
| [in] | bytes | Number of bytes to write. |

### ◆ callbackSet

fsp_err_t(* i2c_slave_api_t::callbackSet) (i2c_slave_ctrl_t *const p_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IIC Slave control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* i2c_slave_api_t::close) (i2c_slave_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Slave device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block set in i2c_slave_api_t::open call. |

### ◆ i2c_slave_instance_t

struct i2c_slave_instance_t

| This structure encompasses everything that is needed to use an instance of this interface. | | |
|---|---|---|
| Data Fields | | |
| i2c_slave_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| i2c_slave_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i2c_slave_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ i2c_slave_ctrl_t

| typedef void i2c_slave_ctrl_t |
|---|
| I2C control block. Allocate an instance specific control block to pass into the I2C API calls. |

### Enumeration Type Documentation

#### ◆ i2c_slave_rate_t

| enum i2c_slave_rate_t | |
|---|---|
| Communication speed options | |
| Enumerator | |
| I2C_SLAVE_RATE_STANDARD | 100 kHz |
| I2C_SLAVE_RATE_FAST | 400 kHz |
| I2C_SLAVE_RATE_FASTPLUS | 1 MHz |
| I2C_SLAVE_RATE_HIGHSPEED | 3.4 MHz |

#### ◆ i2c_slave_addr_mode_t

| enum i2c_slave_addr_mode_t | |
|---|---|
| Addressing mode options | |
| Enumerator | |
| I2C_SLAVE_ADDR_MODE_7BIT | Use 7-bit addressing mode. |
| I2C_SLAVE_ADDR_MODE_10BIT | Use 10-bit addressing mode. |

### ◆ i2c_slave_event_t

| enum i2c_slave_event_t | |
|---|---|
| Callback events | |
| Enumerator | |
| I2C_SLAVE_EVENT_ABORTED | A transfer was aborted. |
| I2C_SLAVE_EVENT_RX_COMPLETE | A receive operation was completed successfully. |
| I2C_SLAVE_EVENT_TX_COMPLETE | A transmit operation was completed successfully. |
| I2C_SLAVE_EVENT_RX_REQUEST | A read operation expected from slave. Detected a write from master. |
| I2C_SLAVE_EVENT_TX_REQUEST | A write operation expected from slave. Detected a read from master. |
| I2C_SLAVE_EVENT_RX_MORE_REQUEST | A read operation expected from slave. Master sends out more data than configured to be read in slave. |
| I2C_SLAVE_EVENT_TX_MORE_REQUEST | A write operation expected from slave. Master requests more data than configured to be written by slave. |
| I2C_SLAVE_EVENT_GENERAL_CALL | General Call address received from Master. Detected a write from master. |

## 5.3.5.6 I2S Interface

Interfaces » Connectivity

### Detailed Description

Interface for I2S audio communication.

# Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

### Data Structures

| | |
|---|---|
| struct | i2s_callback_args_t |
| struct | i2s_status_t |
| struct | i2s_cfg_t |
| struct | i2s_api_t |
| struct | i2s_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | i2s_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | i2s_pcm_width_t |
| enum | i2s_word_length_t |
| enum | i2s_event_t |
| enum | i2s_mode_t |
| enum | i2s_mute_t |
| enum | i2s_ws_continue_t |
| enum | i2s_state_t |

## Data Structure Documentation

### ◆ i2s_callback_args_t

| struct i2s_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t. |
| i2s_event_t | event | The event can be used to identify what caused the callback (overflow or error). |

### ◆ i2s_status_t

| struct i2s_status_t |
|---|
| I2S status. |

| Data Fields | | |
|---|---|---|
| i2s_state_t | state | Current I2S state. |

### ◆ i2s_cfg_t

| struct i2s_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | channel |
| | |
| i2s_pcm_width_t | pcm_width |
| | Audio PCM data width. |
| | |
| i2s_word_length_t | word_length |
| | Audio word length, bits must be >= i2s_cfg_t::pcm_width bits. |
| | |
| i2s_ws_continue_t | ws_continue |
| | Whether to continue WS transmission during idle state. |
| | |
| i2s_mode_t | operating_mode |
| | Master or slave mode. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | |
| void(* | p_callback )(i2s_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

| | |
|---|---|
| uint8_t | rxi_ipl |
| | Receive interrupt priority. |
| | |
| uint8_t | txi_ipl |
| | Transmit interrupt priority. |
| | |
| uint8_t | idle_err_ipl |
| | Idle/Error interrupt priority. |
| | |
| IRQn_Type | txi_irq |
| | Transmit IRQ number. |
| | |
| IRQn_Type | rxi_irq |
| | Receive IRQ number. |
| | |
| IRQn_Type | int_irq |
| | Idle/Error IRQ number. |

## Field Documentation

### ◆ channel

uint32_t i2s_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

### ◆ p_transfer_tx

transfer_instance_t const* i2s_cfg_t::p_transfer_tx

To use DMA for transmitting link a Transfer instance here. Set to NULL if unused.

### ◆ p_transfer_rx

transfer_instance_t const* i2s_cfg_t::p_transfer_rx

To use DMA for receiving link a Transfer instance here. Set to NULL if unused.

### ◆ p_callback

void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)

Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

void* i2s_cfg_t::p_context

Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

### ◆ i2s_api_t

struct i2s_api_t

I2S functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | stop )(i2s_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | mute )(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable) |
| | |
| fsp_err_t(* | write )(i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | read )(i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | writeRead )(i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | statusGet )(i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(i2s_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(i2s_ctrl_t *const p_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context, |

| | |
|---|---|
| | i2s_callback_args_t *const p_callback_memory) |

## Field Documentation

### ◆ open

fsp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

Initial configuration.

**Precondition**

Peripheral clocks and any required output pins should be configured prior to calling this function.

***Note***

*To reconfigure after calling this function, call i2s_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ stop

fsp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with I2S_EVENT_IDLE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |

### ◆ mute

fsp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

Enable or disable mute.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | mute_enable | Whether to enable or disable mute. |

### ◆ write

fsp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_src | Buffer of PCM samples. Must be 4 byte aligned. |
| [in] | bytes | Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8. |

### ◆ read

fsp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_dest | Buffer to store PCM samples. Must be 4 byte aligned. |
| [in] | bytes | Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes. |

### writeRead

fsp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [in] | p_src | Buffer of PCM samples. Must be 4 byte aligned. |
| [in] | p_dest | Buffer to store PCM samples. Must be 4 byte aligned. |
| [in] | bytes | Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes. |

### statusGet

fsp_err_t(* i2s_api_t::statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)

Get current status and store it in provided pointer p_status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |
| [out] | p_status | Current status of the driver. |

### close

fsp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i2s_api_t::open call for this instance. |

### ◆ callbackSet

| fsp_err_t(* i2s_api_t::callbackSet) (i2s_ctrl_t *const p_ctrl, void(*p_callback)(i2s_callback_args_t *), void *const p_context, i2s_callback_args_t *const p_callback_memory) |
|---|

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the I2S control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ i2s_instance_t

| struct i2s_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| i2s_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| i2s_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i2s_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ i2s_ctrl_t

| typedef void i2s_ctrl_t |
|---|
| I2S control block. Allocate an instance specific control block to pass into the I2S API calls. |

## Enumeration Type Documentation

### ◆ i2s_pcm_width_t

| enum i2s_pcm_width_t | |
|---|---|
| Audio PCM width | |
| Enumerator | |
| I2S_PCM_WIDTH_8_BITS | Using 8-bit PCM. |
| I2S_PCM_WIDTH_16_BITS | Using 16-bit PCM. |
| I2S_PCM_WIDTH_18_BITS | Using 18-bit PCM. |
| I2S_PCM_WIDTH_20_BITS | Using 20-bit PCM. |
| I2S_PCM_WIDTH_22_BITS | Using 22-bit PCM. |
| I2S_PCM_WIDTH_24_BITS | Using 24-bit PCM. |
| I2S_PCM_WIDTH_32_BITS | Using 32-bit PCM. |

### ◆ i2s_word_length_t

| enum i2s_word_length_t | |
|---|---|
| Audio system word length. | |
| Enumerator | |
| I2S_WORD_LENGTH_8_BITS | Using 8-bit system word length. |
| I2S_WORD_LENGTH_16_BITS | Using 16-bit system word length. |
| I2S_WORD_LENGTH_24_BITS | Using 24-bit system word length. |
| I2S_WORD_LENGTH_32_BITS | Using 32-bit system word length. |
| I2S_WORD_LENGTH_48_BITS | Using 48-bit system word length. |
| I2S_WORD_LENGTH_64_BITS | Using 64-bit system word length. |
| I2S_WORD_LENGTH_128_BITS | Using 128-bit system word length. |
| I2S_WORD_LENGTH_256_BITS | Using 256-bit system word length. |

### ◆ i2s_event_t

| enum i2s_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| I2S_EVENT_IDLE | Communication is idle. |
| I2S_EVENT_TX_EMPTY | Transmit buffer is below FIFO trigger level. |
| I2S_EVENT_RX_FULL | Receive buffer is above FIFO trigger level. |

### ◆ i2s_mode_t

| enum i2s_mode_t | |
|---|---|
| I2S communication mode | |
| Enumerator | |
| I2S_MODE_SLAVE | Slave mode. |
| I2S_MODE_MASTER | Master mode. |

### ◆ i2s_mute_t

| enum i2s_mute_t | |
|---|---|
| Mute audio samples. | |
| Enumerator | |
| I2S_MUTE_OFF | Disable mute. |
| I2S_MUTE_ON | Enable mute. |

### ◆ i2s_ws_continue_t

| enum i2s_ws_continue_t | |
|---|---|
| Whether to continue WS (word select line) transmission during idle state. | |
| Enumerator | |
| I2S_WS_CONTINUE_ON | Enable WS continue mode. |
| I2S_WS_CONTINUE_OFF | Disable WS continue mode. |

### ◆ i2s_state_t

| enum i2s_state_t | |
|---|---|
| Possible status values returned by i2s_api_t::statusGet. | |
| Enumerator | |
| I2S_STATE_IN_USE | I2S is in use. |
| I2S_STATE_STOPPED | I2S is stopped. |

### 5.3.5.7 I3C Interface

Interfaces » Connectivity

#### Detailed Description

Interface for I3C.

## Summary

The I3C interface provides APIs and definitions for I3C communication.

#### Data Structures

| | |
|---|---|
| struct | i3c_device_status_t |
| struct | i3c_slave_info_t |
| struct | i3c_device_table_cfg_t |
| struct | i3c_device_cfg_t |
| struct | i3c_command_descriptor_t |
| struct | i3c_callback_args_t |
| struct | i3c_cfg_t |
| struct | i3c_api_t |
| struct | i3c_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | i3c_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | i3c_common_command_code_t |
| enum | i3c_event_t |
| enum | i3c_device_type_t |
| enum | i3c_device_protocol_t |
| enum | i3c_address_assignment_mode_t |
| enum | i3c_ibi_type_t |

## Data Structure Documentation

### ◆ i3c_device_status_t

| struct i3c_device_status_t | | |
|---|---|---|
| The current status of the slave device (See GETSTATUS in the MIPI I3C Specification v1.0). | | |
| Data Fields | | |
| uint8_t | pending_interrupt | Contains the interrupt number of any pending interrupt, or 0 if no interrupts are pending. |
| uint8_t | vendor_status | Reserved for vendor-specific meaning. |

### ◆ i3c_slave_info_t

| struct i3c_slave_info_t | | |
|---|---|---|
| Device characteristics that define the I3C capabilities of a slave. | | |
| Data Fields | | |
| uint8_t | pid[6] | Provisional ID. |
| union i3c_slave_info_t | __unnamed__ | |
| uint8_t | dcr | Device Characteristics Register. |

### ◆ i3c_device_table_cfg_t

| struct i3c_device_table_cfg_t | | |
|---|---|---|
| Structure for configuring an entry in the device table when the driver is in master mode (See i3c_api_t::masterDeviceTableSet). | | |
| Data Fields | | |
| uint8_t | static_address | I3C Static address / I2C address for this device. |
| uint8_t | dynamic_address | Dynamic address for the |

| | | |
|---|---|---|
| | | device. This address will be assigned during Dynamic Address Assignment. |
| i3c_device_protocol_t | device_protocol | The protocol used to communicate with this device (I3C / I2C Legacy). |
| bool | ibi_accept | Accept or reject IBI requests from this device. |
| bool | master_request_accept | Accept mastership requests from this device. |
| bool | ibi_payload | IBI requests from this device have a data payload.<br><br>Note: When the device is configured using ENTDAA, the ibi_payload will automatically be updated based on the value of BCR. |

### ◆ i3c_device_cfg_t

| struct i3c_device_cfg_t | | |
|---|---|---|
| Structure for configuring a slave address when the driver is in slave mode (See i3c_api_t::deviceCfgSet). | | |
| Data Fields | | |
| uint8_t | static_address | I3C Static address / I2C address for this device. |
| uint8_t | dynamic_address | Dynamic address for this device. Note that the dynamic address will automatically be updated when ENTDAA is completed. |
| i3c_slave_info_t | slave_info | PID, BCR, and DCR registers for the device (Slave mode only). |

### ◆ i3c_command_descriptor_t

| struct i3c_command_descriptor_t | | |
|---|---|---|
| Descriptor for completing CCC/HDR transfers. | | |
| Data Fields | | |
| uint8_t | command_code | Common Command Code / HDR Command Code for the transfer. |
| uint8_t * | p_buffer | Buffer for reading or writing data. |
| uint32_t | length | Length of the data portion of the command. |

| | | |
|---|---|---|
| bool | restart | If true, issue a repeated-start after the transfer is completed. |
| bool | rnw | Set to true if the command type is Direct Get. |

### ◆ i3c_callback_args_t

| struct i3c_callback_args_t | | |
|---|---|---|
| Arguments that are passed to the user callback when an event occurs. | | |
| Data Fields | | |
| i3c_event_t | event | The type of event that has occurred. |
| uint32_t | event_status | Status flags associated with the event. |
| uint32_t | transfer_size | Number of bytes transferred. |
| i3c_slave_info_t const * | p_slave_info | A pointer to the Characteristics Registers read during ENTDAA. |
| uint8_t | dynamic_address | The dynamic address that was assigned to the slave during ENTDAA. |
| i3c_ibi_type_t | ibi_type | The type of IBI that has been received. |
| uint8_t | ibi_address | The address of the device that sent the IBI. |
| uint8_t | command_code | The command code of the received command. |
| void * | p_context | User defined context. |

### ◆ i3c_cfg_t

| struct i3c_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | channel |
| | |
| i3c_device_type_t | device_type |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | Transfer instance for I3C transmit. Set to NULL if unused. More... |
| | |
| | |

| | |
|---|---|
| transfer_instance_t const * | p_transfer_rx |
| | Transfer instance for I3C receive. Set to NULL if unused. |
| | |
| void(* | p_callback )(i3c_callback_args_t const *const p_args) |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | |

## Field Documentation

### ◆ channel

uint32_t i3c_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

### ◆ device_type

i3c_device_type_t i3c_cfg_t::device_type

The type of device.

### ◆ p_transfer_tx

transfer_instance_t const* i3c_cfg_t::p_transfer_tx

Transfer instance for I3C transmit. Set to NULL if unused.

Transfer API support.

### ◆ p_callback

void(* i3c_cfg_t::p_callback) (i3c_callback_args_t const *const p_args)

Pointer to the user callback.

### ◆ p_extend

void const* i3c_cfg_t::p_extend

Pointer to extended configuration.

### ◆ i3c_api_t

struct i3c_api_t

I3C functions implemented at the HAL layer will follow this API.

| **Data Fields** | |
|---|---|
| fsp_err_t(* | open )(i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | enable )(i3c_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | deviceCfgSet )(i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg) |
| | |
| fsp_err_t(* | masterDeviceTableSet )(i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg) |
| | |
| fsp_err_t(* | deviceSelect )(i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode) |
| | |
| fsp_err_t(* | dynamicAddressAssignmentStart )(i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count) |
| | |
| fsp_err_t(* | slaveStatusSet )(i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status) |
| | |
| fsp_err_t(* | commandSend )(i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t const *const p_command_descriptor) |
| | |
| fsp_err_t(* | write )(i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart) |
| | |
| fsp_err_t(* | read )(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart) |
| | |
| fsp_err_t(* | ibiWrite )(i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length) |
| | |
| fsp_err_t(* | ibiRead )(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length) |
| | |
| fsp_err_t(* | close )(i3c_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* i3c_api_t::open) (i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ enable

fsp_err_t(* i3c_api_t::enable) (i3c_ctrl_t *const p_ctrl)

Enable the I3C device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |

### ◆ deviceCfgSet

fsp_err_t(* i3c_api_t::deviceCfgSet) (i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg)

Set the configuration of this device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | p_device_cfg | Pointer to device configuration. |

### ◆ masterDeviceTableSet

fsp_err_t(* i3c_api_t::masterDeviceTableSet) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)

Set the configuration for the device at the given index in the device table. The configuration will be used by transfers when it is selected by deviceSelect.

Note: This function is not used in slave mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | device_index | Index into the device table. |
| [in] | p_device_table_cfg | Pointer to the table settings for the entry in the master device table. |

### ◆ deviceSelect

fsp_err_t(* i3c_api_t::deviceSelect) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode)

In master mode, select the device for the next transfer.

Note: This function is not used in slave mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | device_index | Index into the device table. |
| [in] | bitrate_setting | The bitrate settings for the selected device. |

### ◆ dynamicAddressAssignmentStart

fsp_err_t(* i3c_api_t::dynamicAddressAssignmentStart) (i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)

Start Dynamic Address Assignment by sending either the ENTDAA or SETDASA command See i3c_address_assignment_mode_t for more information.

Note: This function is not used in slave mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | address_assignment_mode | The command to use for Dynamic Address Assignment. |
| [in] | starting_device_index | The device index that will be used to assign the first device during Dynamic Address Assignment. |
| [in] | device_count | The number of devices to assign (Only used with I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA). |

### ◆ slaveStatusSet

fsp_err_t(* i3c_api_t::slaveStatusSet) (i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status)

Set the status returned to the master in response to a GETSTATUS command.

Note: This function is not used in master mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | device_status | New status settings for responding to the GETSTATUS command code. |

### ◆ commandSend

fsp_err_t(* i3c_api_t::commandSend) (i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t const *const p_command_descriptor)

Send a read/write/broadcast CCC or HDR command.

Note: This function is not used in slave mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | p_command_descriptor | A descriptor for executing the command. |

### ◆ write

fsp_err_t(* i3c_api_t::write) (i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)

In master mode: Start a write transfer. When the transfer is completed send a stop condition or a repeated-start. In slave mode: Set the write buffer and configure the number of bytes that will be transferred before the the transfer is ended by the slave via the 'T' bit or by the master issuing a stop condition.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | p_data | Pointer to a buffer to write. |
| [in] | length | Number of bytes to transfer. |
| [in] | restart | If true, issue a repeated-start after the transfer is completed (Master only). |

### ◆ read

fsp_err_t(* i3c_api_t::read) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart)

In master mode: Start a read transfer. When the transfer is completed, send a stop condition or a repeated-start. In slave mode: Set the read buffer for storing data read during the transfer. When the buffer is full, the application will receive a callback requesting a new read buffer. If no buffer is provided by the application, the driver will discard any remaining bytes read during the transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | p_data | Pointer to a buffer to store the bytes read during the transfer. |
| [in] | length | Number of bytes to transfer. |
| [in] | restart | If true, issue a repeated-start after the transfer is completed (Master only). |

### ◆ ibiWrite

fsp_err_t(* i3c_api_t::ibiWrite) (i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)

Initiate an IBI write operation.

Note: This function is not used in master mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | ibi_type | The type of In-Band Interrupt. |
| [in] | p_data | Pointer to a buffer to start the bytes read during the transfer. |
| [in] | length | Number of bytes to transfer. |

### ◆ ibiRead

| fsp_err_t(* i3c_api_t::ibiRead) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length) |
|---|

Set the read buffer for storing received IBI data (This function is not used in slave mode).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |
| [in] | p_data | Pointer to a buffer to store the bytes read during the transfer. |
| [in] | length | Number of bytes to transfer. |

### ◆ close

| fsp_err_t(* i3c_api_t::close) (i3c_ctrl_t *const p_ctrl) |
|---|

Allows driver to be reconfigured and may reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in i3c_api_t::open call for this instance. |

### ◆ i3c_instance_t

| struct i3c_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| i3c_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| i3c_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| i3c_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ i3c_ctrl_t

| typedef void i3c_ctrl_t |
|---|
| I3C control block. Allocate an instance specific control block to pass into the I3C API calls. |

## Enumeration Type Documentation

### ◆ i3c_common_command_code_t

| enum i3c_common_command_code_t | |
|---|---|
| Common Command Codes defined by MIPI I3C Specification v1.1. | |
| Enumerator | |
| I3C_CCC_BROADCAST_ENEC | Enable Slave initiated events. |
| I3C_CCC_BROADCAST_DISEC | Disable Slave initiated events. |
| I3C_CCC_BROADCAST_ENTAS0 | Enter Activity State 0. |
| I3C_CCC_BROADCAST_ENTAS1 | Enter Activity State 1. |
| I3C_CCC_BROADCAST_ENTAS2 | Enter Activity State 2. |
| I3C_CCC_BROADCAST_ENTAS3 | Enter Activity State 3. |
| I3C_CCC_BROADCAST_RSTDAA | Reset Dynamic Address Assignment. |
| I3C_CCC_BROADCAST_ENTDAA | Enter Dynamic Address Assignment. |
| I3C_CCC_BROADCAST_DEFSVLS | Define List of Slaves. |
| I3C_CCC_BROADCAST_SETMWL | Set Max Write Length. |
| I3C_CCC_BROADCAST_SETMRL | Set Max Read Length. |
| I3C_CCC_BROADCAST_ENTTM | Enter Test Mode. |
| I3C_CCC_BROADCAST_SETBUSCON | Set BUS Context. |
| I3C_CCC_BROADCAST_ENDXFER | Data Transfer Ending Procedure Control. |
| I3C_CCC_BROADCAST_ENTHDR0 | Enter HDR Mode 0. |
| I3C_CCC_BROADCAST_ENTHDR1 | Enter HDR Mode 1. |
| I3C_CCC_BROADCAST_ENTHDR2 | Enter HDR Mode 2. |
| I3C_CCC_BROADCAST_ENTHDR3 | Enter HDR Mode 3. |
| I3C_CCC_BROADCAST_ENTHDR4 | Enter HDR Mode 4 (Reserved for future definition). |
| I3C_CCC_BROADCAST_ENTHDR5 | Enter HDR Mode 5 (Reserved for future definition). |

| | |
|---|---|
| I3C_CCC_BROADCAST_ENTHDR6 | Enter HDR Mode 6 (Reserved for future definition). |
| I3C_CCC_BROADCAST_ENTHDR7 | Enter HDR Mode 7 (Reserved for future definition). |
| I3C_CCC_BROADCAST_SETXTIME | Set Exchange Timing Info. |
| I3C_CCC_BROADCAST_SETAASA | Set All Addresses to Static Address. |
| I3C_CCC_BROADCAST_RSTACT | Slave Reset Action. |
| I3C_CCC_BROADCAST_DEFGRPA | Define List of Group Address. |
| I3C_CCC_BROADCAST_RSTGRPA | Reset Group Address. |
| I3C_CCC_BROADCAST_MLANE | Multi-Lane Data Transfer Control. |
| I3C_CCC_DIRECT_ENEC | Enable Slave initiated events. |
| I3C_CCC_DIRECT_DISEC | Disable Slave initiated events. |
| I3C_CCC_DIRECT_ENTAS0 | Enter Activity State 0. |
| I3C_CCC_DIRECT_ENTAS1 | Enter Activity State 1. |
| I3C_CCC_DIRECT_ENTAS2 | Enter Activity State 2. |
| I3C_CCC_DIRECT_ENTAS3 | Enter Activity State 3. |
| I3C_CCC_DIRECT_RSTDAA | Reset Dynamic Address Assignment (DEPRECATED v1.0). |
| I3C_CCC_DIRECT_SETDASA | Set Dynamic Address from Static Address. |
| I3C_CCC_DIRECT_SETNEWDA | Set New Dynamic Address. |
| I3C_CCC_DIRECT_SETMWL | Set Max Write Length. |
| I3C_CCC_DIRECT_SETMRL | Set Max Read Length. |
| I3C_CCC_DIRECT_GETMWL | Get Max Write Length. |
| I3C_CCC_DIRECT_GETMRL | Get Max Read Length. |
| I3C_CCC_DIRECT_GETPID | Get Provisional ID. |
| I3C_CCC_DIRECT_GETBCR | Get Bus Characteristic Register. |

| | |
|---|---|
| I3C_CCC_DIRECT_GETDCR | Get Device Characteristic Register. |
| I3C_CCC_DIRECT_GETSTATUS | Get Device Status. |
| I3C_CCC_DIRECT_GETACCMST | Get Accept Mastership. |
| I3C_CCC_DIRECT_ENDXFER | Data Transfer Ending Procedure Control. |
| I3C_CCC_DIRECT_SETBRGTGT | Set Bridge Targets. |
| I3C_CCC_DIRECT_GETMXDS | Get Max Data Speed. |
| I3C_CCC_DIRECT_GETHDRCAP | Get HDR Capability. |
| I3C_CCC_DIRECT_SETROUTE | Set Route. |
| I3C_CCC_DIRECT_D2DXFER | Device to Device(s) Tunneling Control. |
| I3C_CCC_DIRECT_SETXTIME | Set Exchange Timing Information. |
| I3C_CCC_DIRECT_GETXTIME | Get Exchange Timing Information. |
| I3C_CCC_DIRECT_RSTACT | Reset Slave Action. |
| I3C_CCC_DIRECT_SETGRPA | Set Group Address. |
| I3C_CCC_DIRECT_RSTGRPA | Reset Group Address. |
| I3C_CCC_DIRECT_MLANE | Multi-Lane Data Transfer Control. |

### ◆ i3c_event_t

| enum i3c_event_t | |
|---|---|
| I3C Events that result in a callback. | |
| Enumerator | |
| I3C_EVENT_ENTDAA_ADDRESS_PHASE | Events that only occur in Master mode. A Slave device has finished writing its PID, BCR, and DCR. This information is provided in i3c_callback_args_t::p_slave_info. |
| I3C_EVENT_IBI_READ_COMPLETE | An IBI has successfully been read. |
| I3C_EVENT_IBI_READ_BUFFER_FULL | There is no more space in the IBI read buffer. The application may provide another buffer by calling i3c_api_t::ibiRead. |
| I3C_EVENT_READ_BUFFER_FULL | Events that only occur in Slave mode. There is no more space in the read buffer. The application may provide another buffer by calling i3c_api_t::read. |
| I3C_EVENT_IBI_WRITE_COMPLETE | A IBI was written successfully. |
| I3C_EVENT_HDR_EXIT_PATTERN_DETECTED | The HDR exit pattern was detected on the bus. |
| I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE | Dynamic Address Assignment has completed.<br>Events that are common to Master and Slave mode. |
| I3C_EVENT_COMMAND_COMPLETE | A command was completed. |
| I3C_EVENT_WRITE_COMPLETE | A write transfer has completed. |
| I3C_EVENT_READ_COMPLETE | A read transfer has completed. |
| I3C_EVENT_TIMEOUT_DETECTED | SCL is stuck at the logic high or logic low level during a transfer. |
| I3C_EVENT_INTERNAL_ERROR | An internal error occurred. |
| I3C_EVENT_SDA_WRITE_COMPLETE | An SDA (Short Data Argument) write transfer has completed. |

### ◆ i3c_device_type_t

| enum i3c_device_type_t | |
|---|---|
| The type of device. | |
| Enumerator | |
| I3C_DEVICE_TYPE_MAIN_MASTER | The main master starts in master mode and is responsible for configuring the bus. |
| I3C_DEVICE_TYPE_SLAVE | A slave device listens to the bus for relevant I3C Commands (CCCs) sent by the current master, and responds accordingly. Slave devices may also initiate In-band interrupts and Hot-Join requests. |

### ◆ i3c_device_protocol_t

| enum i3c_device_protocol_t | |
|---|---|
| Identifies the protocol for transferring data with the device on the bus. | |
| Enumerator | |
| I3C_DEVICE_PROTOCOL_I2C | Transfers will use legacy I2C protocol with open-drain output at a reduced baudrate. |
| I3C_DEVICE_PROTOCOL_I3C | Transfers will use I3C SDR mode. |

### ◆ i3c_address_assignment_mode_t

| enum i3c_address_assignment_mode_t | |
|---|---|
| Address Assignment Mode. | |
| Enumerator | |
| I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA | Send the ENTDAA command to enter Dynamic Address Assignment mode and assign dynamic addresses in order, starting with the starting device index. The procedure is completed after the specified number of devices have been configured. The callback will be called after the PID, DCR, and BCR registers have been read for each device. |

### ◆ i3c_ibi_type_t

| enum i3c_ibi_type_t | |
|---|---|
| The type of In-Band Interrupt. | |
| Enumerator | |
| I3C_IBI_TYPE_INTERRUPT | Application specific In-Band Interrupt for notifying the master when an event occurs. |
| I3C_IBI_TYPE_HOT_JOIN | Request the master to perform the Dynamic Address Assignment process. |
| I3C_IBI_TYPE_MASTERSHIP_REQUEST | Request the master to give up control of the bus. |

### 5.3.5.8 LIN Interface

Interfaces » Connectivity

#### Detailed Description

Interface for LIN communications.

# Summary

The LIN interface provides common APIs for LIN HAL drivers. The LIN interface supports the following features:

- Half-duplex master or slave LIN communication
- Interrupt driven transmit/receive processing
- Callback function with returned event code and data
- Checksum generation and validation (standard or enhanced)
- Bus Sleep and Wakeup

#### Data Structures

| | |
|---|---|
| struct | lin_transfer_params_t |
| struct | lin_callback_args_t |
| struct | lin_cfg_t |
| struct | lin_api_t |
| struct | lin_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | lin_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | lin_mode_t |
| enum | lin_checksum_type_t |
| enum | lin_event_t |

## Data Structure Documentation

### ◆ lin_transfer_params_t

| struct lin_transfer_params_t | | |
|---|---|---|
| LIN Transfer Parameters | | |
| Data Fields | | |
| uint8_t | id | The unprotected frame ID associated with the data transfer. |
| union lin_transfer_params_t | __unnamed__ | |
| uint8_t | num_bytes | Length of buffer pointed to by p_data, in bytes. |
| lin_checksum_type_t | checksum_type | Checksum type to use for checksum generation (when writing data) or validation (when reading data). See lin_checksum_type_t. |

### ◆ lin_callback_args_t

| struct lin_callback_args_t | | |
|---|---|---|
| LIN Callback Arguments | | |
| Data Fields | | |
| uint32_t | channel | Channel number. |
| lin_event_t | event | Event code. |
| uint8_t | bytes_received | Valid for the following events:<br>• LIN_EVENT_RX_DATA_COMPLETE<br>• LIN_EVENT_ERR_FRAMING<br>• LIN_EVENT_ERR_INVALID_CHECKSUM<br>Contains the number of data |

| | | |
|---|---|---|
| | | bytes received for a data reception. |
| uint8_t | pid | For LIN slave: Contains the most recently received protected identifier For LIN master: Contains the most recently transmitted protected identifier. |
| uint8_t | checksum | Received checksum. Valid for the following events:<br>• LIN_EVENT_RX_DATA_COMPLETE<br>• LIN_EVENT_ERR_INVALID_CHECKSUM. |
| void * | p_context | Context provided to user during callback. |

### ◆ lin_cfg_t

| struct lin_cfg_t | |
|---|---|
| LIN configuration block | |
| **Data Fields** | |
| uint8_t | channel |
| | Select a channel corresponding to the channel number of the hardware. |
| | |
| lin_mode_t | mode |
| | Driver mode (master or slave) |
| | |
| void(* | p_callback )(lin_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | LIN hardware dependent configuration. |

### ◆ lin_api_t

| struct lin_api_t | |
|---|---|
| Interface definition for LIN | |
| **Data Fields** | |
| fsp_err_t(* | open )(lin_ctrl_t *const p_ctrl, lin_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(lin_ctrl_t *const p_ctrl, const lin_transfer_params_t *const p_transfer_params) |
| | |
| fsp_err_t(* | read )(lin_ctrl_t *const p_ctrl, lin_transfer_params_t *const p_transfer_params) |
| | |
| fsp_err_t(* | communicationAbort )(lin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(lin_ctrl_t *const p_ctrl, void(*p_callback)(lin_callback_args_t *), void *const p_context, lin_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | wakeupSend )(lin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | sleepEnter )(lin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | sleepExit )(lin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(lin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | informationFrameRead )(lin_ctrl_t *const p_ctrl, lin_transfer_params_t *const p_transfer_params) |
| | |
| fsp_err_t(* | startFrameWrite )(lin_ctrl_t *const p_ctrl, uint8_t const id) |
| | |
| fsp_err_t(* | informationFrameWrite )(lin_ctrl_t *const p_ctrl, const lin_transfer_params_t *const p_transfer_params) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* lin_api_t::open) (lin_ctrl_t *const p_ctrl, lin_cfg_t const *const p_cfg)

Open LIN device. Transmission and reception of LIN frames is enabled upon successful return from this function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the LIN control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to LIN configuration structure. All elements of this structure must be set by user. |

### ◆ write

fsp_err_t(* lin_api_t::write) (lin_ctrl_t *const p_ctrl, const lin_transfer_params_t *const p_transfer_params)

Begin non-blocking transmission of a LIN frame.

If the node is a LIN master, the LIN header is transmitted before data, if present.

For header-only transmissions (master mode), when the header transmission completes successfully, the callback is called with the event LIN_EVENT_TX_HEADER_COMPLETE.

For header+data (master mode) or data-only transmission (slave mode), when the data transmission completes successfully, the callback is called with event LIN_EVENT_TX_DATA_COMPLETE.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the LIN control block. |
| [in] | p_transfer_params | Pointer to parameters required for the write transaction. |

### ◆ read

fsp_err_t(* lin_api_t::read) (lin_ctrl_t *const p_ctrl, lin_transfer_params_t *const p_transfer_params)

Begin non-blocking read of data bytes.

When a read completes successfully, the callback is called with event LIN_EVENT_RX_DATA_COMPLETE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block for the channel. |
| [in] | p_transfer_params | Pointer to parameters required for the read transaction. |

### ◆ communicationAbort

fsp_err_t(* lin_api_t::communicationAbort) (lin_ctrl_t *const p_ctrl)

Abort ongoing transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |

### ◆ callbackSet

fsp_err_t(* lin_api_t::callbackSet) (lin_ctrl_t *const p_ctrl, void(*p_callback)(lin_callback_args_t *), void *const p_context, lin_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ wakeupSend

fsp_err_t(* lin_api_t::wakeupSend) (lin_ctrl_t *const p_ctrl)

Send wakeup signal for LIN device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |

### ◆ sleepEnter

fsp_err_t(* lin_api_t::sleepEnter) (lin_ctrl_t *const p_ctrl)

Place the LIN node in bus sleep mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |

### ◆ sleepExit

fsp_err_t(* lin_api_t::sleepExit) (lin_ctrl_t *const p_ctrl)

Exit the bus sleep mode for LIN device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |

### ◆ close

fsp_err_t(* lin_api_t::close) (lin_ctrl_t *const p_ctrl)

Close LIN device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LIN control block. |

### ◆ informationFrameRead

fsp_err_t(* lin_api_t::informationFrameRead) (lin_ctrl_t *const p_ctrl, lin_transfer_params_t *const p_transfer_params)

[DEPRECATED] Use lin_api_t::read

◆ **startFrameWrite**

| fsp_err_t(* lin_api_t::startFrameWrite) (lin_ctrl_t *const p_ctrl, uint8_t const id) |
|---|
| [DEPRECATED] Use lin_api_t::write |

◆ **informationFrameWrite**

| fsp_err_t(* lin_api_t::informationFrameWrite) (lin_ctrl_t *const p_ctrl, const lin_transfer_params_t *const p_transfer_params) |
|---|
| [DEPRECATED] Use lin_api_t::write |

◆ **lin_instance_t**

| struct lin_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| lin_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| lin_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| lin_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

◆ **lin_ctrl_t**

| typedef void lin_ctrl_t |
|---|
| LIN control block. Allocate an instance specific control block to pass into the LIN API calls. |

## Enumeration Type Documentation

◆ **lin_mode_t**

| enum lin_mode_t | |
|---|---|
| LIN driver mode | |
| Enumerator | |
| LIN_MODE_SLAVE | Slave mode. |
| LIN_MODE_MASTER | Master mode. |

### ◆ lin_checksum_type_t

| enum lin_checksum_type_t | |
|---|---|
| LIN checksum type | |
| Enumerator | |
| LIN_CHECKSUM_TYPE_CLASSIC | 8 bit LIN classic checksum over data bytes only |
| LIN_CHECKSUM_TYPE_ENHANCED | 8 bit LIN enhanced checksum over data bytes and PID |
| LIN_CHECKSUM_TYPE_NONE | Skip driver checksum generation/validation. |

### ◆ lin_event_t

| enum lin_event_t | |
|---|---|
| LIN Event codes | |
| Enumerator | |
| LIN_EVENT_NONE | No event present. |
| LIN_EVENT_RX_START_FRAME_COMPLETE | DEPRECATED, do not use. |
| LIN_EVENT_RX_INFORMATION_FRAME_COMPLETE | DEPRECATED, do not use. |
| LIN_EVENT_TX_START_FRAME_COMPLETE | DEPRECATED, do not use. |
| LIN_EVENT_TX_INFORMATION_FRAME_COMPLETE | DEPRECATED, do not use. |
| LIN_EVENT_RX_HEADER_COMPLETE | Header received event. |
| LIN_EVENT_RX_DATA_COMPLETE | Data received event. |
| LIN_EVENT_TX_HEADER_COMPLETE | Header transmission complete event. |
| LIN_EVENT_TX_DATA_COMPLETE | Data transmission complete event. |
| LIN_EVENT_ERR_INVALID_CHECKSUM | Data received successfully, but checksum was invalid. |
| LIN_EVENT_TX_WAKEUP_COMPLETE | Transmit wake up complete event. |
| LIN_EVENT_RX_WAKEUP_COMPLETE | Receive wake up complete event. |
| LIN_EVENT_ERR_BUS_COLLISION_DETECTED | Bus collision detection event. |
| LIN_EVENT_ERR_COUNTER_OVERFLOW | Counter overflow event. |
| LIN_EVENT_ERR_OVERRUN | Overrun error event. |
| LIN_EVENT_ERR_PARITY | Parity error event (header only, LIN data is sent without parity) |
| LIN_EVENT_ERR_FRAMING | Framing error event. |

## 5.3.5.9 SMCI Interface

Interfaces » Connectivity

### Detailed Description

Interface for SMCI communications.

# Summary

The SMCI interface provides common APIs for SMCI HAL drivers. The SMCI interface supports the following features:

- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change (baud = 1/ETU)
- Hardware resource locking during a transaction

### Data Structures

| | |
|---|---|
| struct | smci_status_t |
| struct | smci_transfer_mode_t |
| struct | smci_speed_params_t |
| struct | smci_callback_args_t |
| struct | smci_cfg_t |
| struct | smci_api_t |
| struct | smci_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | smci_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | smci_state_t |
| enum | smci_event_t |
| enum | smci_convention_type_t |
| enum | smci_clock_conversion_integer_t |
| enum | smci_baudrate_adjustment_integer_t |
| enum | smci_protocol_type_t |

## Data Structure Documentation

### ◆ smci_status_t

| struct smci_status_t | | |
|---|---|---|
| SMCI driver specific information | | |
| Data Fields | | |
| smci_state_t | smci_state | State of the SMCI state machine. |
| uint32_t | bytes_recvd | Bytes read into receive buffer since read was called. |

### ◆ smci_transfer_mode_t

| struct smci_transfer_mode_t | | |
|---|---|---|
| SMCI Transfer Mode settings | | |
| Data Fields | | |
| smci_protocol_type_t | protocol | Protocol (Normal t=0, or Block t=1) |
| smci_convention_type_t | convention | Convention Direct or Inverse. |
| bool | gsm_mode | True=GMS Mode, false=Normal. |

### ◆ smci_speed_params_t

| struct smci_speed_params_t | | |
|---|---|---|
| SMCI settings that are used as inputs to register setting calculations | | |
| Data Fields | | |
| uint32_t | baudrate | Bits per second requested, 1/ETU. |
| smci_baudrate_adjustment_integer_t | di | Referred to as D in ISO spec (from Table 8 in ISO7816-3 3rd Edition) |
| smci_clock_conversion_integer_t | fi | Index of F in ISO spec (from Table 7 in ISO7816-3 3rd Edition) |

### ◆ smci_callback_args_t

| struct smci_callback_args_t | | |
|---|---|---|
| SMCI Callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| smci_event_t | event | Event code. |

| | | |
|---|---|---|
| uint8_t | data | Data Byte to process.<br><br>Contains the next character received for the events SMCI_EVENT_RX_CHAR, SMCI_EVENT_ERR_PARITY, SMCI_EVENT_ERR_LOW_SIGNAL, or SMCI_EVENT_ERR_OVERRUN. Otherwise unused. |
| void * | p_context | Context provided to user during callback. |

### ◆ smci_cfg_t

| struct smci_cfg_t | |
|---|---|
| Configuration Structure for SMCI | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel number of the hardware. |
| | |
| uint8_t | rxi_ipl |
| | Receive interrupt priority. |
| | |
| uint8_t | txi_ipl |
| | Transmit interrupt priority. |
| | |
| uint8_t | eri_ipl |
| | Error interrupt priority. |
| | |
| IRQn_Type | rxi_irq |
| | Receive interrupt IRQ number. |
| | |
| IRQn_Type | txi_irq |
| | Transmit interrupt IRQ number. |
| | |

| | |
|---|---|
| IRQn_Type | eri_irq |
| | Error interrupt IRQ number. |
| | |
| void(* | p_callback )(smci_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | SMCI hardware dependent configuration. |
| | |

### ◆ smci_api_t

| struct smci_api_t | |
|---|---|
| Shared Interface definition for SMCI | |
| **Data Fields** | |
| fsp_err_t(* | open )(smci_ctrl_t *const p_ctrl, smci_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(smci_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(smci_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | transferModeSet )(smci_ctrl_t *const p_ctrl, smci_transfer_mode_t const *const p_transfer_mode_params) |
| | |
| fsp_err_t(* | baudSet )(smci_ctrl_t *const p_ctrl, void const *const p_baud_setting) |
| | |
| fsp_err_t(* | statusGet )(smci_ctrl_t *const p_ctrl, smci_status_t *const p_status) |
| | |
| fsp_err_t(* | clockControl )(smci_ctrl_t *const p_ctrl, bool clock_enable) |

| | |
|---|---|
| fsp_err_t(* | callbackSet )(smci_ctrl_t *const p_ctrl, void(*p_callback)(smci_callback_args_t *), void *const p_context, smci_callback_args_t *const p_callback_memory) |
| fsp_err_t(* | close )(smci_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* smci_api_t::open) (smci_ctrl_t *const p_ctrl, smci_cfg_t const *const p_cfg)

Open Smart Card Interface Mode (SMCI)

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the SMCI control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to SMCI configuration structure. All elements of this structure must be set by user. |

### ◆ read

fsp_err_t(* smci_api_t::read) (smci_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read from Smart Card device. The read buffer is used until the read is complete. When a transfer is complete, the callback is called with event SMCI_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event SMCI_EVENT_RX_CHAR.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block for the channel. |
| [in] | p_dest | Destination address to read data from. |
| [in] | bytes | Read data length. |

### ◆ write

fsp_err_t(* smci_api_t::write) (smci_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

Write to Smart Card device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event SMCI_EVENT_TX_COMPLETE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [in] | p_src | Source address to write data to. |
| [in] | bytes | Write data length. |

### ◆ transferModeSet

fsp_err_t(* smci_api_t::transferModeSet) (smci_ctrl_t *const p_ctrl, smci_transfer_mode_t const *const p_transfer_mode_params)

Change the peripheral settings based on provided transfer mode and data convention type

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [in] | p_transfer_mode_params | Pointer to SMCI setting like protocol, convention, and gsm_mode |

### ◆ baudSet

fsp_err_t(* smci_api_t::baudSet) (smci_ctrl_t *const p_ctrl, void const *const p_baud_setting)

Change baud rate.

**Warning**

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [in] | p_baud_setting | Pointer to module specific setting for configuring baud rate. |

### ◆ statusGet

fsp_err_t(* smci_api_t::statusGet) (smci_ctrl_t *const p_ctrl, smci_status_t *const p_status)

Get the driver specific information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [out] | p_status | State info for the driver. |

### ◆ clockControl

fsp_err_t(* smci_api_t::clockControl) (smci_ctrl_t *const p_ctrl, bool clock_enable)

Enable or disable the SMCI clock to control the start of the activation or de-activation

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [in] | clock_enable | True: enables clock output, False disables it |

### ◆ callbackSet

fsp_err_t(* smci_api_t::callbackSet) (smci_ctrl_t *const p_ctrl, void(*p_callback)(smci_callback_args_t *), void *const p_context, smci_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and callback memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SMCI control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

| fsp_err_t(* smci_api_t::close) (smci_ctrl_t *const p_ctrl) |
|---|
| Close SMCI device.<br><br>**Parameters**<br>[in] p_ctrl — Pointer to the SMCI control block. |

### ◆ smci_instance_t

| struct smci_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| smci_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| smci_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| smci_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ smci_ctrl_t

| typedef void smci_ctrl_t |
|---|
| Smart Card Interface control block. Allocate an instance specific control block to pass into the SMCI API calls. |

## Enumeration Type Documentation

### ◆ smci_state_t

| enum smci_state_t | |
|---|---|
| Enumerator | |
| SMCI_STATE_IDLE_CLOCK_OFF | SMCI idle state with no clock output. |
| SMCI_STATE_TX_RX_IDLE | SMCI is in idle state, clock is active. |
| SMCI_STATE_TX_PROGRESSING | Transmission is in progress. |
| SMCI_STATE_RX_PROGRESSING | Reception is in progress. |

### ◆ smci_event_t

| enum smci_event_t | |
|---|---|
| SMCI Event codes | |
| Enumerator | |
| SMCI_EVENT_RX_COMPLETE | Receive complete event. |
| SMCI_EVENT_TX_COMPLETE | Transmit complete event. |
| SMCI_EVENT_RX_CHAR | Character transfer is completed. |
| SMCI_EVENT_ERR_PARITY | Parity error event. |
| SMCI_EVENT_ERR_LOW_SIGNAL | Low error signal response occurred event. |
| SMCI_EVENT_ERR_OVERRUN | Overrun error event. |

### ◆ smci_convention_type_t

| enum smci_convention_type_t | |
|---|---|
| Enumerator | |
| SMCI_CONVENTION_TYPE_DIRECT | Direct convention type (LSB First, High=1) |
| SMCI_CONVENTION_TYPE_INVERSE | Inverse convention type (MSB First, Low=1) |

### ◆ smci_clock_conversion_integer_t

| enum smci_clock_conversion_integer_t | |
|---|---|
| Enumerator | |
| SMCI_CLOCK_CONVERSION_INTEGER_372_4 | Fi = 372, max freq = 4MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_372_5 | Fi = 372, max freq = 5MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_558_6 | Fi = 558, max freq = 6MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_744_8 | Fi = 744, max freq = 8MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_1116_12 | Fi = 1116, max freq = 12MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_1488_16 | Fi = 1488, max freq = 16MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_1860_20 | Fi = 1860, max freq = 20MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_UNSUPPORTED7 | Unsupported. |
| SMCI_CLOCK_CONVERSION_INTEGER_UNSUPPORTED8 | Unsupported. |
| SMCI_CLOCK_CONVERSION_INTEGER_512_5 | Fi = 512, max freq = 5MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_768_75 | Fi = 768, max freq = 7.5MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_1024_10 | Fi = 1024, max freq = 10MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_1536_15 | Fi = 1536, max freq = 15MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_2048_20 | Fi = 2048, max freq = 20MHz. |
| SMCI_CLOCK_CONVERSION_INTEGER_UNSUPPORTED14 | Unsupported. |
| SMCI_CLOCK_CONVERSION_INTEGER_UNSUPPORTED15 | Unsupported. |

### ◆ smci_baudrate_adjustment_integer_t

| enum smci_baudrate_adjustment_integer_t | |
|---|---|
| Enumerator | |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU0 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_1 | Di=1. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_2 | Di=2. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_4 | Di=4. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_8 | Di=8. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_16 | Di=16. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_32 | Di=32. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_64 | Di=64. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_12 | Di=12. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_20 | Di=20. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU10 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU11 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU12 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU13 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU14 | RESERVED. |
| SMCI_BAUDRATE_ADJUSTMENT_INTEGER_RFU15 | RESERVED. |

### ◆ smci_protocol_type_t

| enum smci_protocol_type_t | |
|---|---|
| SMCI Protocol Type according to ISO7816-3 | |
| Enumerator | |
| SMCI_PROTOCOL_TYPE_T0 | Normal mode operation (Protocol T = 0) |
| SMCI_PROTOCOL_TYPE_T1 | Block transfer mode operation (Protocol T = 1) |

## 5.3.5.10 SPI Interface

Interfaces » Connectivity

### Detailed Description

Interface for SPI communications.

# Summary

Provides a common interface for communication using the SPI Protocol.

### Data Structures

| | |
|---|---|
| struct | spi_callback_args_t |
| struct | spi_write_read_guard_args_t |
| struct | spi_cfg_t |
| struct | spi_api_t |
| struct | spi_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | spi_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_bit_width_t |
| enum | spi_mode_t |
| enum | spi_clk_phase_t |

| | |
|---|---|
| enum | spi_clk_polarity_t |
| enum | spi_mode_fault_t |
| enum | spi_bit_order_t |
| enum | spi_event_t |

## Data Structure Documentation

### ◆ spi_callback_args_t

| struct spi_callback_args_t | | |
|---|---|---|
| Common callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| spi_event_t | event | Event code. |
| void * | p_context | Context provided to user during callback. |

### ◆ spi_write_read_guard_args_t

| struct spi_write_read_guard_args_t |
|---|
| Non-secure arguments for write-read guard function |

### ◆ spi_cfg_t

| struct spi_cfg_t | |
|---|---|
| SPI interface configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel number to be used. |
| | |
| IRQn_Type | rxi_irq |
| | Receive Buffer Full IRQ number. |
| | |
| IRQn_Type | txi_irq |
| | Transmit Buffer Empty IRQ number. |
| | |

| | |
|---|---|
| IRQn_Type | tei_irq |
| | Transfer Complete IRQ number. |
| | |
| IRQn_Type | eri_irq |
| | Error IRQ number. |
| | |
| uint8_t | rxi_ipl |
| | Receive Interrupt priority. |
| | |
| uint8_t | txi_ipl |
| | Transmit Interrupt priority. |
| | |
| uint8_t | tei_ipl |
| | Transfer Complete Interrupt priority. |
| | |
| uint8_t | eri_ipl |
| | Error Interrupt priority. |
| | |
| spi_mode_t | operating_mode |
| | Select master or slave operating mode. |
| | |
| spi_clk_phase_t | clk_phase |
| | Data sampling on odd or even clock edge. |
| | |
| spi_clk_polarity_t | clk_polarity |
| | Clock level when idle. |
| | |
| spi_mode_fault_t | mode_fault |

| | |
|---|---|
| | Mode fault error (master/slave conflict) flag. |
| | |
| spi_bit_order_t | bit_order |
| | Select to transmit MSB/LSB first. |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | To use SPI DTC/DMAC write transfer, link a transfer instance here. Set to NULL if unused. |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | To use SPI DTC/DMAC read transfer, link a transfer instance here. Set to NULL if unused. |
| | |
| void(* | p_callback )(spi_callback_args_t *p_args) |
| | Pointer to user callback function. |
| | |
| void * | p_context |
| | User defined context passed to callback function. |
| | |
| void const * | p_extend |
| | Extended SPI hardware dependent configuration. |
| | |

◆ **spi_api_t**

| struct spi_api_t | |
|---|---|
| Shared Interface definition for SPI | |
| **Data Fields** | |
| fsp_err_t(* | open )(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |

| | |
|---|---|
| | |
| fsp_err_t(* | write )(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width) |
| | |
| fsp_err_t(* | writeRead )(spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width) |
| | |
| fsp_err_t(* | callbackSet )(spi_ctrl_t *const p_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(spi_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to user-provided storage for the control block. |
| [in] | p_cfg | Pointer to SPI configuration structure. |

### ◆ read

fsp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Receive data from a SPI device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [out] | p_dest | Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |

### ◆ write

fsp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

Transmit data to a SPI device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [in] | p_src | Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |

### ◆ writeRead

fsp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Simultaneously transmit data to a SPI device while receiving data from a SPI device (full duplex).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |
| [in] | p_src | Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL. |
| [out] | p_dest | Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL. |
| [in] | length | Number of units of data to be transferred (unit size specified by the bit_width). |
| [in] | bit_width | Data bit width to be transferred. |

### ◆ callbackSet

fsp_err_t(* spi_api_t::callbackSet) (spi_ctrl_t *const p_ctrl, void(*p_callback)(spi_callback_args_t *), void *const p_context, spi_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the SPI control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control block for the channel. |

### ◆ spi_instance_t

struct spi_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| spi_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| spi_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| spi_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ spi_ctrl_t

| typedef void spi_ctrl_t |
| --- |
| SPI control block. Allocate an instance specific control block to pass into the SPI API calls. |

## Enumeration Type Documentation

### ◆ spi_bit_width_t

| enum spi_bit_width_t | |
| --- | --- |
| Data bit width | |
| Enumerator | |
| SPI_BIT_WIDTH_4_BITS | Data bit width is 4 bits. |
| SPI_BIT_WIDTH_5_BITS | Data bit width is 5 bits. |
| SPI_BIT_WIDTH_6_BITS | Data bit width is 6 bits. |
| SPI_BIT_WIDTH_7_BITS | Data bit width is 7 bits. |
| SPI_BIT_WIDTH_8_BITS | Data bit width is 8 bits. |
| SPI_BIT_WIDTH_9_BITS | Data bit width is 9 bits. |
| SPI_BIT_WIDTH_10_BITS | Data bit width is 10 bits. |
| SPI_BIT_WIDTH_11_BITS | Data bit width is 11 bits. |
| SPI_BIT_WIDTH_12_BITS | Data bit width is 12 bits. |
| SPI_BIT_WIDTH_13_BITS | Data bit width is 13 bits. |
| SPI_BIT_WIDTH_14_BITS | Data bit width is 14 bits. |
| SPI_BIT_WIDTH_15_BITS | Data bit width is 15 bits. |
| SPI_BIT_WIDTH_16_BITS | Data bit width is 16 bits. |
| SPI_BIT_WIDTH_17_BITS | Data bit width is 17 bits. |
| SPI_BIT_WIDTH_18_BITS | Data bit width is 18 bits. |
| SPI_BIT_WIDTH_19_BITS | Data bit width is 19 bits. |
| SPI_BIT_WIDTH_20_BITS | Data bit width is 20 bits. |

| | |
|---|---|
| SPI_BIT_WIDTH_21_BITS | Data bit width is 21 bits. |
| SPI_BIT_WIDTH_22_BITS | Data bit width is 22 bits. |
| SPI_BIT_WIDTH_23_BITS | Data bit width is 23 bits. |
| SPI_BIT_WIDTH_24_BITS | Data bit width is 24 bits. |
| SPI_BIT_WIDTH_25_BITS | Data bit width is 25 bits. |
| SPI_BIT_WIDTH_26_BITS | Data bit width is 26 bits. |
| SPI_BIT_WIDTH_27_BITS | Data bit width is 27 bits. |
| SPI_BIT_WIDTH_28_BITS | Data bit width is 28 bits. |
| SPI_BIT_WIDTH_29_BITS | Data bit width is 29 bits. |
| SPI_BIT_WIDTH_30_BITS | Data bit width is 30 bits. |
| SPI_BIT_WIDTH_31_BITS | Data bit width is 31 bits. |
| SPI_BIT_WIDTH_32_BITS | Data bit width is 32 bits. |

### ◆ spi_mode_t

| enum spi_mode_t | |
|---|---|
| Master or slave operating mode | |
| Enumerator | |
| SPI_MODE_MASTER | Channel operates as SPI master. |
| SPI_MODE_SLAVE | Channel operates as SPI slave. |

### ◆ spi_clk_phase_t

| enum spi_clk_phase_t | |
|---|---|
| Clock phase | |
| Enumerator | |
| SPI_CLK_PHASE_EDGE_ODD | 0: Data sampling on odd edge, data variation on even edge |
| SPI_CLK_PHASE_EDGE_EVEN | 1: Data variation on odd edge, data sampling on even edge |

### ◆ spi_clk_polarity_t

| enum spi_clk_polarity_t | |
|---|---|
| Clock polarity | |
| Enumerator | |
| SPI_CLK_POLARITY_LOW | 0: Clock polarity is low when idle |
| SPI_CLK_POLARITY_HIGH | 1: Clock polarity is high when idle |

### ◆ spi_mode_fault_t

| enum spi_mode_fault_t | |
|---|---|
| Mode fault error flag. This error occurs when the device is setup as a master, but the SSLA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave. | |
| Enumerator | |
| SPI_MODE_FAULT_ERROR_ENABLE | Mode fault error flag on. |
| SPI_MODE_FAULT_ERROR_DISABLE | Mode fault error flag off. |

### ◆ spi_bit_order_t

| enum spi_bit_order_t | |
|---|---|
| Bit order | |
| Enumerator | |
| SPI_BIT_ORDER_MSB_FIRST | Send MSB first in transmission. |
| SPI_BIT_ORDER_LSB_FIRST | Send LSB first in transmission. |

### ◆ spi_event_t

| enum spi_event_t | |
|---|---|
| SPI events | |
| Enumerator | |
| SPI_EVENT_TRANSFER_COMPLETE | The data transfer was completed. |
| SPI_EVENT_TRANSFER_ABORTED | The data transfer was aborted. |
| SPI_EVENT_ERR_MODE_FAULT | Mode fault error. |
| SPI_EVENT_ERR_READ_OVERFLOW | Read overflow error. |
| SPI_EVENT_ERR_PARITY | Parity error. |
| SPI_EVENT_ERR_OVERRUN | Overrun error. |
| SPI_EVENT_ERR_FRAMING | Framing error. |
| SPI_EVENT_ERR_MODE_UNDERRUN | Underrun error. |

## 5.3.5.11 UART Interface

Interfaces » Connectivity

### Detailed Description

Interface for UART communications.

# Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)

### Data Structures

| | |
|---|---|
| struct | uart_info_t |
| struct | uart_callback_args_t |
| struct | uart_cfg_t |
| struct | uart_api_t |
| struct | uart_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | uart_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | uart_event_t |
| enum | uart_data_bits_t |
| enum | uart_parity_t |
| enum | uart_stop_bits_t |
| enum | uart_dir_t |

## Data Structure Documentation

### ◆ uart_info_t

| struct uart_info_t | | |
|---|---|---|
| UART driver specific information | | |
| Data Fields | | |
| uint32_t | write_bytes_max | Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL. |

| | | |
|---|---|---|
| uint32_t | read_bytes_max | Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL. |

### ◆ uart_callback_args_t

| struct uart_callback_args_t | | |
|---|---|---|
| UART Callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| uart_event_t | event | Event code. |
| uint32_t | data | Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused. |
| void * | p_context | Context provided to user during callback. |

### ◆ uart_cfg_t

| struct uart_cfg_t | |
|---|---|
| UART Configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Select a channel corresponding to the channel number of the hardware. |
| | |
| uart_data_bits_t | data_bits |
| | Data bit length (8 or 7 or 9) |
| | |
| uart_parity_t | parity |
| | Parity type (none or odd or even) |
| | |
| uart_stop_bits_t | stop_bits |
| | Stop bit length (1 or 2) |

| | |
|---|---|
| | |
| uint8_t | rxi_ipl |
| | Receive interrupt priority. |
| | |
| IRQn_Type | rxi_irq |
| | Receive interrupt IRQ number. |
| | |
| uint8_t | txi_ipl |
| | Transmit interrupt priority. |
| | |
| IRQn_Type | txi_irq |
| | Transmit interrupt IRQ number. |
| | |
| uint8_t | tei_ipl |
| | Transmit end interrupt priority. |
| | |
| IRQn_Type | tei_irq |
| | Transmit end interrupt IRQ number. |
| | |
| uint8_t | eri_ipl |
| | Error interrupt priority. |
| | |
| IRQn_Type | eri_irq |
| | Error interrupt IRQ number. |
| | |
| transfer_instance_t const * | p_transfer_rx |
| | |
| transfer_instance_t const * | p_transfer_tx |
| | |

| | |
|---|---|
| void(* | p_callback )(uart_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | UART hardware dependent configuration. |
| | |

## Field Documentation

### ◆ p_transfer_rx

transfer_instance_t const* uart_cfg_t::p_transfer_rx

Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

### ◆ p_transfer_tx

transfer_instance_t const* uart_cfg_t::p_transfer_tx

Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

### ◆ uart_api_t

struct uart_api_t

Shared Interface definition for UART

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | write )(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes) |
| | |
| fsp_err_t(* | baudSet )(uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info) |

| | |
|---|---|
| | |
| fsp_err_t(* | infoGet )(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info) |
| | |
| fsp_err_t(* | communicationAbort )(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort) |
| | |
| fsp_err_t(* | callbackSet )(uart_ctrl_t *const p_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(uart_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | readStop )(uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes) |
| | |
| fsp_err_t(* | receiveSuspend )(uart_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | receiveResume )(uart_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

Open UART device.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the UART control block. Must be declared by user. Value set here. |
| [in] | uart_cfg_t | Pointer to UART configuration structure. All elements of this structure must be set by user. |

### ◆ read

fsp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read from UART device. The read buffer is used until the read is complete. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet().

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block for the channel. |
| [in] | p_dest | Destination address to read data from. |
| [in] | bytes | Read data length. |

### ◆ write

fsp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet().

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | p_src | Source address to write data to. |
| [in] | bytes | Write data length. |

### ◆ baudSet

fsp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)

Change baud rate.

**Warning**

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | p_baudrate_info | Pointer to module specific information for configuring baud rate. |

### ◆ infoGet

fsp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

Get the driver specific information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [out] | p_info | Pointer to UART information structure. |

### ◆ communicationAbort

fsp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Abort ongoing transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | communication_to_abort | Type of abort request. |

### ◆ callbackSet

fsp_err_t(* uart_api_t::callbackSet) (uart_ctrl_t *const p_ctrl, void(*p_callback)(uart_callback_args_t *), void *const p_context, uart_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)

Close UART device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |

### ◆ readStop

fsp_err_t(* uart_api_t::readStop) (uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes)

Stop ongoing read and return the number of bytes remaining in the read.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |
| [in,out] | remaining_bytes | Pointer to location to store remaining bytes for read. |

### ◆ receiveSuspend

| fsp_err_t(* uart_api_t::receiveSuspend) (uart_ctrl_t *const p_ctrl) |
|---|

Suspend RX operations for UART device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |

### ◆ receiveResume

| fsp_err_t(* uart_api_t::receiveResume) (uart_ctrl_t *const p_ctrl) |
|---|

Resume RX operations for UART device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the UART control block. |

### ◆ uart_instance_t

| struct uart_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| uart_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| uart_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| uart_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ uart_ctrl_t

| typedef void uart_ctrl_t |
|---|

UART control block. Allocate an instance specific control block to pass into the UART API calls.

## Enumeration Type Documentation

### ◆ uart_event_t

| enum uart_event_t | |
|---|---|
| UART Event codes | |
| Enumerator | |
| UART_EVENT_RX_COMPLETE | Receive complete event. |
| UART_EVENT_TX_COMPLETE | Transmit complete event. |
| UART_EVENT_RX_CHAR | Character received. |
| UART_EVENT_ERR_PARITY | Parity error event. |
| UART_EVENT_ERR_FRAMING | Mode fault error event. |
| UART_EVENT_ERR_OVERFLOW | FIFO Overflow error event. |
| UART_EVENT_BREAK_DETECT | Break detect error event. |
| UART_EVENT_TX_DATA_EMPTY | Last byte is transmitting, ready for more data. |

### ◆ uart_data_bits_t

| enum uart_data_bits_t | |
|---|---|
| UART Data bit length definition | |
| Enumerator | |
| UART_DATA_BITS_5 | Data bits 5-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_5 | Data bits 5-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_5 | Data bits 5-bit. |

| | |
|---|---|
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_5 | Data bits 5-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_5 | Data bits 5-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_5 | Data bits 5-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_9 | Data bits 9-bit. |
| UART_DATA_BITS_8 | Data bits 8-bit. |
| UART_DATA_BITS_7 | Data bits 7-bit. |

### ◆ uart_parity_t

| enum uart_parity_t | |
|---|---|
| UART Parity definition | |
| Enumerator | |
| UART_PARITY_OFF | No parity. |
| UART_PARITY_ZERO | Zero parity. |
| UART_PARITY_EVEN | Even parity. |
| UART_PARITY_ODD | Odd parity. |

### ◆ uart_stop_bits_t

| enum uart_stop_bits_t | |
|---|---|
| UART Stop bits definition | |
| Enumerator | |
| UART_STOP_BITS_1 | Stop bit 1-bit. |
| UART_STOP_BITS_2 | Stop bits 2-bit. |

### ◆ uart_dir_t

| enum uart_dir_t | |
|---|---|
| UART transaction definition | |
| Enumerator | |
| UART_DIR_RX_TX | Both RX and TX. |
| UART_DIR_RX | Only RX. |
| UART_DIR_TX | Only TX. |

## 5.3.5.12 USB HAUD Interface

Interfaces » Connectivity

### Detailed Description

Interface for USB HAUD functions.

# Summary

The USB HAUD interface provides USB HAUD functionality.

### Data Structures

| | |
|---|---|
| struct | usb_haud_sampling_t |
| struct | usb_haud_device_info_t |
| struct | usb_haud_api_t |

### Data Structure Documentation

#### ◆ usb_haud_sampling_t

| struct usb_haud_sampling_t | | |
|---|---|---|
| Sampling Frequency Settings | | |
| Data Fields | | |
| uint32_t | channels | User Input Parameter : Number of channels. |
| uint32_t | frequency | User Input Parameter : Sampling frequency. |
| uint32_t | resolution | User Input Parameter : bits per sample. |
| uint8_t | frequency_setting | Whether the frequency needs to be set to device. |
| uint8_t | alternate_setting | Whether the ALT number is set. |
| uint32_t | interface_number | Set by HAUD Driver : Interface number. |
| uint32_t | alternate_number | Set by HAUD Driver : Alternate number. |
| uint8_t * | p_sampling_descriptor | Set by HAUD Driver : Pointer to Audio Streaming Format Type Descriptor (Audio1.0) |
| uint16_t | max_packet_size | Set by HAUD Driver : Max Packet Size. |
| uint8_t | endpoint_address | Set by HAUD Driver : Endpoint Address. |

#### ◆ usb_haud_device_info_t

| struct usb_haud_device_info_t | | |
|---|---|---|
| Audio protocol data | | |
| Data Fields | | |
| uint8_t | protocol | Protocol code. |

### ◆ usb_haud_api_t

| struct usb_haud_api_t | |
|---|---|
| USB HAUD functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | infoGet )(usb_ctrl_t *const p_ctrl, usb_haud_device_info_t *p_info, uint8_t device_address) |
| | |
| fsp_err_t(* | alternateNumberGet )(usb_ctrl_t *const p_ctrl, usb_haud_sampling_t *p_info, uint8_t direction, uint8_t device_address) |
| | |
| fsp_err_t(* | pipeSet )(usb_ctrl_t *const p_ctrl, uint32_t interface_number, uint32_t alternate_number, uint8_t device_address) |
| | |

## Field Documentation

### ◆ infoGet

fsp_err_t(* usb_haud_api_t::infoGet) (usb_ctrl_t *const p_ctrl, usb_haud_device_info_t *p_info, uint8_t device_address)

Get connected device information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to store Audio device information. |
| [in] | device_address | Device address. |

### ◆ alternateNumberGet

fsp_err_t(* usb_haud_api_t::alternateNumberGet) (usb_ctrl_t *const p_ctrl, usb_haud_sampling_t *p_info, uint8_t direction, uint8_t device_address)

Get alternate number and related information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to store Audio device information. |
| [in] | direction | Streaming data direction. |
| [in] | device_address | Device address. |

### ◆ pipeSet

fsp_err_t(* usb_haud_api_t::pipeSet) (usb_ctrl_t *const p_ctrl, uint32_t interface_number, uint32_t alternate_number, uint8_t device_address)

Pipe Setting.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | interface_number | Interface number used in SET_INTERFACE. |
| [in] | alternate_number | Alternate number used in SET_INTERFACE. |
| [in] | device_address | Device address. |

## 5.3.5.13 USB HCDC Interface

Interfaces » Connectivity

### Detailed Description

Interface for USB HCDC functions.

# Summary

The USB HCDC interface provides USB HCDC functionality.

### Data Structures

| | |
|---|---|
| struct | usb_hcdc_encapsulated_t |

| | |
|---|---|
| struct | usb_hcdc_abstractstate_t |
| struct | usb_hcdc_countrysetting_t |
| union | usb_hcdc_commfeature_t |
| struct | usb_hcdc_linecoding_t |
| struct | usb_hcdc_controllinestate_t |
| struct | usb_hcdc_serialstate_t |
| struct | usb_hcdc_breakduration_t |
| struct | usb_hcdc_device_info_t |
| struct | usb_hcdc_api_t |
| struct | usb_hcdc_instance_t |

### Enumerations

| | |
|---|---|
| enum | usb_hcdc_data_bit_t |
| enum | usb_hcdc_stop_bit_t |
| enum | usb_hcdc_parity_bit_t |
| enum | usb_hcdc_line_speed_t |
| enum | usb_hcdc_feature_selector_t |

## Data Structure Documentation

### ◆ usb_hcdc_encapsulated_t

| struct usb_hcdc_encapsulated_t | | |
|---|---|---|
| Encapsulated data | | |
| Data Fields | | |
| uint8_t * | p_data | Protocol dependent data. |
| uint16_t | wlength | Data length in bytes. |

### ◆ usb_hcdc_abstractstate_t

| struct usb_hcdc_abstractstate_t |
|---|
| Abstract Control Model (ACM) settings bitmap |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | bis: 1 | Idle enable. |
| uint16_t | bdms: 1 | Data multiplexing enable. |
| uint16_t | rsv: 14 | Reserved. |

### ◆ usb_hcdc_countrysetting_t

| struct usb_hcdc_countrysetting_t | | |
|---|---|---|
| Country code data | | |
| Data Fields | | |
| uint16_t | country_code | Country code. |

### ◆ usb_hcdc_commfeature_t

| union usb_hcdc_commfeature_t | | |
|---|---|---|
| Feature setting data | | |
| Data Fields | | |
| usb_hcdc_abstractstate_t | abstract_state | ACM settings bitmap. |
| usb_hcdc_countrysetting_t | country_setting | Country code. |

### ◆ usb_hcdc_linecoding_t

| struct usb_hcdc_linecoding_t | | |
|---|---|---|
| Virtual UART configuration (line coding) | | |
| Data Fields | | |
| usb_hcdc_line_speed_t | dwdte_rate | Data terminal rate in bits per second. |
| usb_hcdc_stop_bit_t | bchar_format | Stop bits. |
| usb_hcdc_parity_bit_t | bparity_type | Parity. |
| usb_hcdc_data_bit_t | bdata_bits | Data bits. |
| uint8_t | rsv | Reserved. |

### ◆ usb_hcdc_controllinestate_t

| struct usb_hcdc_controllinestate_t | | |
|---|---|---|
| Virtual UART control signal bitmap | | |
| Data Fields | | |
| uint16_t | bdtr: 1 | DTR. |
| uint16_t | brts: 1 | RTS. |
| uint16_t | rsv: 14 | Reserved. |

### ◆ usb_hcdc_serialstate_t

| struct usb_hcdc_serialstate_t |
|---|

| Virtual UART state bitmap | | |
|---|---|---|
| Data Fields | | |
| uint16_t | brx_carrier: 1 | DCD signal. |
| uint16_t | btx_carrier: 1 | DSR signal. |
| uint16_t | bbreak: 1 | Break detection status. |
| uint16_t | bring_signal: 1 | Ring signal. |
| uint16_t | bframing: 1 | Framing error. |
| uint16_t | bparity: 1 | Parity error. |
| uint16_t | bover_run: 1 | Over Run error. |
| uint16_t | rsv: 9 | Reserved. |

### ◆ usb_hcdc_breakduration_t

| struct usb_hcdc_breakduration_t | | |
|---|---|---|
| Break duration data | | |
| Data Fields | | |
| uint16_t | wtime_ms | Duration of Break. |

### ◆ usb_hcdc_device_info_t

| struct usb_hcdc_device_info_t | | |
|---|---|---|
| Break duration data | | |
| Data Fields | | |
| uint16_t | vendor_id | Vendor ID. |
| uint16_t | product_id | Product ID. |
| uint8_t | subclass | Subclass code. |

### ◆ usb_hcdc_api_t

| struct usb_hcdc_api_t | |
|---|---|
| USB HCDC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | controlDataRead )(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address) |
| | |
| fsp_err_t(* | deviceRegister )(usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id) |
| | |
| fsp_err_t(* | infoGet )(usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address) |

## Field Documentation

### ◆ controlDataRead

fsp_err_t(* usb_hcdc_api_t::controlDataRead) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t device_address)

Read Control Data (CDC Interrupt IN data)

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores read data. |
| [in] | size | Read request size. |
| [in] | device_address | Device address. |

### ◆ deviceRegister

fsp_err_t(* usb_hcdc_api_t::deviceRegister) (usb_ctrl_t *const p_api_ctrl, uint16_t vendor_id, uint16_t product_id)

Register the specified vendor class device in the device table.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | vendor_id | Vendor ID. |
| [in] | product_id | Product ID. |

### ◆ infoGet

fsp_err_t(* usb_hcdc_api_t::infoGet) (usb_ctrl_t *const p_api_ctrl, usb_hcdc_device_info_t *p_info, uint8_t device_address)

Get connected device information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to store CDC device information. |
| [in] | device_address | Device address. |

### ◆ usb_hcdc_instance_t

struct usb_hcdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| usb_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| usb_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| usb_hcdc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Enumeration Type Documentation

### ◆ usb_hcdc_data_bit_t

enum usb_hcdc_data_bit_t

Virtual UART data length

| Enumerator | |
|---|---|
| USB_HCDC_DATA_BIT_7 | 7 bits |
| USB_HCDC_DATA_BIT_8 | 8 bits |

### ◆ usb_hcdc_stop_bit_t

enum usb_hcdc_stop_bit_t

Virtual UART stop bit length

| Enumerator | |
|---|---|
| USB_HCDC_STOP_BIT_1 | 1 bit |
| USB_HCDC_STOP_BIT_15 | 1.5 bits |
| USB_HCDC_STOP_BIT_2 | 2 bits |

### ◆ usb_hcdc_parity_bit_t

| enum usb_hcdc_parity_bit_t | |
|---|---|
| Virtual UART parity bit setting | |
| Enumerator | |
| USB_HCDC_PARITY_BIT_NONE | No parity bit. |
| USB_HCDC_PARITY_BIT_ODD | Odd parity. |
| USB_HCDC_PARITY_BIT_EVEN | Even parity. |

### ◆ usb_hcdc_line_speed_t

| enum usb_hcdc_line_speed_t |
|---|
| Virtual UART bitrate |

### ◆ usb_hcdc_feature_selector_t

| enum usb_hcdc_feature_selector_t |
|---|
| Feature Selector |

## 5.3.5.14 USB HHID Interface

Interfaces » Connectivity

### Detailed Description

Interface for USB HHID functions.

# Summary

The USB HHID interface provides USB HHID functionality.

### Data Structures

| | |
|---|---|
| struct | usb_hhid_api_t |
| struct | usb_hhid_instance_t |

### Macros

| | |
|---|---|
| #define | USB_HID_OTHER<br>Other. |
| #define | USB_HID_KEYBOARD<br>Keyboard. |
| #define | USB_HID_MOUSE<br>Mouse. |
| #define | USB_HID_IN<br>In Transfer. |
| #define | USB_HID_OUT<br>Out Transfer. |

## Data Structure Documentation

### ◆ usb_hhid_api_t

| struct usb_hhid_api_t | |
|---|---|
| USB HHID functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | typeGet )(usb_ctrl_t *const p_ctrl, uint8_t *p_type, uint8_t device_address) |
| | |
| fsp_err_t(* | maxPacketSizeGet )(usb_ctrl_t *const p_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address) |
| | |

## Field Documentation

### ◆ typeGet

fsp_err_t(* usb_hhid_api_t::typeGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_type, uint8_t device_address)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_type | Pointer to store HID protocol value. |
| [in] | device_address | Device Address. |

### ◆ maxPacketSizeGet

fsp_err_t(* usb_hhid_api_t::maxPacketSizeGet) (usb_ctrl_t *const p_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_size | Pointer to the area to store the max package size. |
| [in] | direction | Transfer direction. |
| [in] | device_address | Device Address. |

### ◆ usb_hhid_instance_t

struct usb_hhid_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| usb_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| usb_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| usb_hhid_api_t const * | p_api | Pointer to the API structure for this instance. |

## 5.3.5.15 USB HMSC Interface

Interfaces » Connectivity

### Detailed Description

Interface for USB HMSC functions.

# Summary

The USB HMSC interface provides USB HMSC functionality.

### Data Structures

| | |
|---|---|
| struct | usb_hmsc_api_t |

### Enumerations

| | |
|---|---|
| enum | usb_atapi_t |
| enum | usb_csw_result_t |

### Data Structure Documentation

#### ◆ usb_hmsc_api_t

| struct usb_hmsc_api_t | |
|---|---|
| USB HMSC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | storageCommand )(usb_ctrl_t *const p_ctrl, uint8_t *buf, uint8_t command, uint8_t destination) |
| | |
| fsp_err_t(* | driveNumberGet )(usb_ctrl_t *const p_ctrl, uint8_t *p_drive, uint8_t destination) |
| | |
| fsp_err_t(* | storageReadSector )(uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count) |
| | |
| fsp_err_t(* | storageWriteSector )(uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count) |
| | |
| fsp_err_t(* | semaphoreGet )(void) |
| | |
| fsp_err_t(* | semaphoreRelease )(void) |
| | |

## Field Documentation

### ◆ storageCommand

fsp_err_t(* usb_hmsc_api_t::storageCommand) (usb_ctrl_t *const p_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)

Processing for MassStorage(ATAPI) command.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | *buf | Pointer to the buffer area to store the transfer data. |
| [in] | command | ATAPI command. |
| [in] | destination | Represents a device address. |

### ◆ driveNumberGet

fsp_err_t(* usb_hmsc_api_t::driveNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_drive, uint8_t destination)

Get number of Storage drive.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_drive | Store address for Drive No. |
| [in] | destination | Represents a device address. |

### ◆ storageReadSector

fsp_err_t(* usb_hmsc_api_t::storageReadSector) (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)

Read sector information.

**Parameters**

| | | |
|---|---|---|
| [in] | drive_number | Drive number. |
| [out] | *buff | Pointer to the buffer area to store the transfer data. |
| [in] | sector_number | The sector number to start with. |
| [in] | sector_count | Transmit with the sector size of the number of times. |

### ◆ storageWriteSector

fsp_err_t(* usb_hmsc_api_t::storageWriteSector) (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)

Write sector information.

**Parameters**

| | | |
|---|---|---|
| [in] | drive_number | Drive number. |
| [in] | *buff | Pointer to the buffer area to store the transfer data. |
| [in] | sector_number | The sector number to start with. |
| [in] | sector_count | Transmit with the sector size of the number of times. |

### ◆ semaphoreGet

fsp_err_t(* usb_hmsc_api_t::semaphoreGet) (void)

Get Semaphore.

### ◆ semaphoreRelease

fsp_err_t(* usb_hmsc_api_t::semaphoreRelease) (void)

Release Semaphore.

## Enumeration Type Documentation

### ◆ usb_atapi_t

| enum usb_atapi_t | |
|---|---|
| ATAPI commands | |
| Enumerator | |
| USB_ATAPI_TEST_UNIT_READY | Test Unit Ready. |
| USB_ATAPI_REQUEST_SENSE | Request Sense. |
| USB_ATAPI_FORMAT_UNIT | Format Unit. |
| USB_ATAPI_INQUIRY | Inquiry. |
| USB_ATAPI_MODE_SELECT6 | Mode Select6. |
| USB_ATAPI_MODE_SENSE6 | Mode Sense6. |
| USB_ATAPI_START_STOP_UNIT | Start Stop Unit. |
| USB_ATAPI_PREVENT_ALLOW | Prevent Allow. |
| USB_ATAPI_READ_FORMAT_CAPACITY | Read Format Capacity. |
| USB_ATAPI_READ_CAPACITY | Read Capacity. |
| USB_ATAPI_READ10 | Read10. |
| USB_ATAPI_WRITE10 | Write10. |
| USB_ATAPI_SEEK | Seek. |
| USB_ATAPI_WRITE_AND_VERIFY | Write and Verify. |
| USB_ATAPI_VERIFY10 | Verify10. |
| USB_ATAPI_MODE_SELECT10 | Mode Select10. |
| USB_ATAPI_MODE_SENSE10 | Mode Sense10. |

### ◆ usb_csw_result_t

| enum usb_csw_result_t | |
|---|---|
| Command Status Wrapper (CSW) | |
| Enumerator | |
| USB_CSW_RESULT_SUCCESS | CSW was successful. |
| USB_CSW_RESULT_FAIL | CSW failed. |
| USB_CSW_RESULT_PHASE | CSW has phase error. |

#### 5.3.5.16 USB Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB functions.

## Summary

The USB interface provides USB functionality.

#### Data Structures

| | |
|---|---|
| struct | usb_cfg_t |
| struct | usb_api_t |
| struct | usb_instance_t |

#### Macros

| | |
|---|---|
| #define | USB_BREQUEST<br>b15-8 |
| #define | USB_GET_STATUS<br>USB Standard request Get Status. |
| #define | USB_CLEAR_FEATURE<br>USB Standard request Clear Feature. |

| | |
|---|---|
| #define | USB_REQRESERVED<br>USB Standard request Reqreserved. |
| #define | USB_SET_FEATURE<br>USB Standard request Set Feature. |
| #define | USB_REQRESERVED1<br>USB Standard request Reqreserved1. |
| #define | USB_SET_ADDRESS<br>USB Standard request Set Address. |
| #define | USB_GET_DESCRIPTOR<br>USB Standard request Get Descriptor. |
| #define | USB_SET_DESCRIPTOR<br>USB Standard request Set Descriptor. |
| #define | USB_GET_CONFIGURATION<br>USB Standard request Get Configuration. |
| #define | USB_SET_CONFIGURATION<br>USB Standard request Set Configuration. |
| #define | USB_GET_INTERFACE<br>USB Standard request Get Interface. |
| #define | USB_SET_INTERFACE<br>USB Standard request Set Interface. |
| #define | USB_SYNCH_FRAME<br>USB Standard request Synch Frame. |

| | |
|---|---|
| #define | USB_HOST_TO_DEV<br>From host to device. |
| #define | USB_DEV_TO_HOST<br>From device to host. |
| #define | USB_STANDARD<br>Standard Request. |
| #define | USB_CLASS<br>Class Request. |
| #define | USB_VENDOR<br>Vendor Request. |
| #define | USB_DEVICE<br>Device. |
| #define | USB_INTERFACE<br>Interface. |
| #define | USB_ENDPOINT<br>End Point. |
| #define | USB_OTHER<br>Other. |
| #define | USB_NULL<br>NULL pointer. |
| #define | USB_IP0<br>USB0 module. |
| #define | USB_IP1 |

| | |
|---|---|
| | USB1 module. |
| #define | USB_PIPE0<br>Pipe Number0. |
| #define | USB_PIPE1<br>Pipe Number1. |
| #define | USB_PIPE2<br>Pipe Number2. |
| #define | USB_PIPE3<br>Pipe Number3. |
| #define | USB_PIPE4<br>Pipe Number4. |
| #define | USB_PIPE5<br>Pipe Number5. |
| #define | USB_PIPE6<br>Pipe Number6. |
| #define | USB_PIPE7<br>Pipe Number7. |
| #define | USB_PIPE8<br>Pipe Number8. |
| #define | USB_PIPE9<br>Pipe Number9. |
| #define | USB_EP0<br>End Point Number0. |

| | |
|---|---|
| #define | USB_EP1<br>End Point Number1. |
| #define | USB_EP2<br>End Point Number2. |
| #define | USB_EP3<br>End Point Number3. |
| #define | USB_EP4<br>End Point Number4. |
| #define | USB_EP5<br>End Point Number5. |
| #define | USB_EP6<br>End Point Number6. |
| #define | USB_EP7<br>End Point Number7. |
| #define | USB_EP8<br>End Point Number8. |
| #define | USB_EP9<br>End Point Number9. |
| #define | USB_EP10<br>End Point Number10. |
| #define | USB_EP11<br>End Point Number11. |

| | |
|---|---|
| #define | USB_EP12<br>End Point Number12. |
| #define | USB_EP13<br>End Point Number13. |
| #define | USB_EP14<br>End Point Number14. |
| #define | USB_EP15<br>End Point Number15. |
| #define | USB_EP_DIR<br>b7: Endpoint Direction |
| #define | USB_EP_DIR_IN<br>b7: Endpoint Direction In |
| #define | USB_EP_DIR_OUT<br>b7: Endpoint Direction Out |
| #define | USB_DT_DEVICE<br>Device Descriptor. |
| #define | USB_DT_CONFIGURATION<br>Configuration Descriptor. |
| #define | USB_DT_STRING<br>String Descriptor. |
| #define | USB_DT_INTERFACE<br>Interface Descriptor. |
| #define | USB_DT_ENDPOINT |

| | |
|---|---|
| | Endpoint Descriptor. |
| #define | USB_DT_DEVICE_QUALIFIER<br>Device Qualifier Descriptor. |
| #define | USB_DT_OTHER_SPEED_CONF<br>Other Speed Configuration Descriptor. |
| #define | USB_DT_INTERFACE_POWER<br>Interface Power Descriptor. |
| #define | USB_DT_OTGDESCRIPTOR<br>OTG Descriptor. |
| #define | USB_DT_HUBDESCRIPTOR<br>HUB descriptor. |
| #define | USB_IFCLS_NOT<br>Un corresponding Class. |
| #define | USB_IFCLS_AUD<br>Audio Class. |
| #define | USB_IFCLS_CDC<br>CDC Class. |
| #define | USB_IFCLS_CDCC<br>CDC-Control Class. |
| #define | USB_IFCLS_HID<br>HID Class. |
| #define | USB_IFCLS_PHY<br>Physical Class. |

| | |
|---|---|
| #define | USB_IFCLS_IMG<br>Image Class. |
| #define | USB_IFCLS_PRN<br>Printer Class. |
| #define | USB_IFCLS_MAS<br>Mass Storage Class. |
| #define | USB_IFCLS_HUB<br>HUB Class. |
| #define | USB_IFCLS_CDCD<br>CDC-Data Class. |
| #define | USB_IFCLS_CHIP<br>Chip/Smart Card Class. |
| #define | USB_IFCLS_CNT<br>Content-Security Class. |
| #define | USB_IFCLS_VID<br>Video Class. |
| #define | USB_IFCLS_DIAG<br>Diagnostic Device. |
| #define | USB_IFCLS_WIRE<br>Wireless Controller. |
| #define | USB_IFCLS_APL<br>Application-Specific. |

| | |
|---|---|
| #define | USB_IFCLS_VEN<br>Vendor-Specific Class. |
| #define | USB_EP_IN<br>In Endpoint. |
| #define | USB_EP_OUT<br>Out Endpoint. |
| #define | USB_EP_ISO<br>Isochronous Transfer. |
| #define | USB_EP_BULK<br>Bulk Transfer. |
| #define | USB_EP_INT<br>Interrupt Transfer. |
| #define | USB_CF_RESERVED<br>Reserved(set to 1) |
| #define | USB_CF_SELFP<br>Self Powered. |
| #define | USB_CF_BUSP<br>Bus Powered. |
| #define | USB_CF_RWUPON<br>Remote Wake up ON. |
| #define | USB_CF_RWUPOFF<br>Remote Wake up OFF. |
| #define | USB_DD_BLENGTH |

| | |
|---|---|
| | Device Descriptor Length. |
| #define | USB_CD_BLENGTH<br>Configuration Descriptor Length. |
| #define | USB_ID_BLENGTH<br>Interface Descriptor Length. |
| #define | USB_ED_BLENGTH<br>Endpoint Descriptor Length. |

### Typedefs

| | |
|---|---|
| typedef void | usb_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | usb_speed_t |
| enum | usb_setup_status_t |
| enum | usb_status_t |
| enum | usb_class_t |
| enum | usb_bcport_t |
| enum | usb_onoff_t |
| enum | usb_transfer_t |
| enum | usb_transfer_type_t |
| enum | usb_mode_t |
| enum | usb_compliancetest_status_t |

## Data Structure Documentation

### ◆ usb_cfg_t

| struct usb_cfg_t |
|---|
| USB configuration. |
| Data Fields |

| | | |
|---|---|---|
| usb_mode_t | usb_mode | USB_MODE_HOST/USB_MODE_PERI. |
| usb_speed_t | usb_speed | USB speed (USB_HS/USB_FS/USB_LS) |
| uint8_t | module_number | USB module number (USB_IP0/USB_IP1) |
| usb_class_t | type | USB device class etc. |
| usb_descriptor_t * | p_usb_reg | Pointer to the usb_decriptor_t structure area. |
| usb_compliance_cb_t * | usb_complience_cb | |
| IRQn_Type | irq | USBI dedicated interrupt number storage variable. |
| IRQn_Type | irq_r | USBR dedicated interrupt number storage variable. |
| IRQn_Type | irq_d0 | FS D0FIFO dedicated interrupt number storage variable. |
| IRQn_Type | irq_d1 | FS D1FIFO dedicated interrupt number storage variable. |
| IRQn_Type | hsirq | USBIR dedicated interrupt number storage variable. |
| IRQn_Type | hsirq_d0 | HS D0FIFO dedicated interrupt number storage variable. |
| IRQn_Type | hsirq_d1 | HS D1FIFO dedicated interrupt number storage variable. |
| uint8_t | ipl | Variable to store the interrupt priority of USBI. |
| uint8_t | ipl_r | Variable to store the interrupt priority of USBR. |
| uint8_t | ipl_d0 | Variable to store the interrupt priority of FS D0FIFO. |
| uint8_t | ipl_d1 | Variable to store the interrupt priority of FS D1FIFO. |
| uint8_t | hsipl | Variable to store the interrupt priority of USBIR. |
| uint8_t | hsipl_d0 | Variable to store the interrupt priority of HS D0FIFO. |
| uint8_t | hsipl_d1 | Variable to store the interrupt priority of HS D1FIFO. |
| usb_callback_t * | p_usb_apl_callback | Application Callback. |
| void * | p_context | Other Context. |
| const transfer_instance_t * | p_transfer_tx | Send context. |

| | | |
|---|---|---|
| const transfer_instance_t * | p_transfer_rx | Receive context. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ usb_api_t

| struct usb_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(usb_ctrl_t *const p_ctrl, usb_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(usb_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination) |
| | |
| fsp_err_t(* | write )(usb_ctrl_t *const p_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination) |
| | |
| fsp_err_t(* | stop )(usb_ctrl_t *const p_ctrl, usb_transfer_t direction, uint8_t destination) |
| | |
| fsp_err_t(* | suspend )(usb_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | resume )(usb_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | vbusSet )(usb_ctrl_t *const p_ctrl, uint16_t state) |
| | |
| fsp_err_t(* | infoGet )(usb_ctrl_t *const p_ctrl, usb_info_t *p_info, uint8_t destination) |
| | |
| fsp_err_t(* | pipeRead )(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) |
| | |
| fsp_err_t(* | pipeWrite )(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) |
| | |
| | |

| | |
|---|---|
| fsp_err_t(* | pipeStop )(usb_ctrl_t *const p_ctrl, uint8_t pipe_number) |
| | |
| fsp_err_t(* | usedPipesGet )(usb_ctrl_t *const p_ctrl, uint16_t *p_pipe, uint8_t destination) |
| | |
| fsp_err_t(* | pipeInfoGet )(usb_ctrl_t *const p_ctrl, usb_pipe_t *p_info, uint8_t pipe_number) |
| | |
| fsp_err_t(* | eventGet )(usb_ctrl_t *const p_ctrl, usb_status_t *event) |
| | |
| fsp_err_t(* | callback )(usb_callback_t *p_callback) |
| | |
| fsp_err_t(* | pullUp )(usb_ctrl_t *const p_ctrl, uint8_t state) |
| | |
| fsp_err_t(* | hostControlTransfer )(usb_ctrl_t *const p_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address) |
| | |
| fsp_err_t(* | periControlDataGet )(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size) |
| | |
| fsp_err_t(* | periControlDataSet )(usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size) |
| | |
| fsp_err_t(* | periControlStatusSet )(usb_ctrl_t *const p_ctrl, usb_setup_status_t status) |
| | |
| fsp_err_t(* | remoteWakeup )(usb_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | driverActivate )(usb_ctrl_t *const p_api_ctrl) |
| | |
| fsp_err_t(* | callbackMemorySet )(usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory) |
| | |
| fsp_err_t(* | moduleNumberGet )(usb_ctrl_t *const p_ctrl, uint8_t *module_number) |
| | |
| fsp_err_t(* | classTypeGet )(usb_ctrl_t *const p_ctrl, usb_class_t *class_type) |

| | |
|---|---|
| fsp_err_t(* | deviceAddressGet )(usb_ctrl_t *const p_ctrl, uint8_t *device_address) |
| fsp_err_t(* | pipeNumberGet )(usb_ctrl_t *const p_ctrl, uint8_t *pipe_number) |
| fsp_err_t(* | deviceStateGet )(usb_ctrl_t *const p_ctrl, uint16_t *state) |
| fsp_err_t(* | dataSizeGet )(usb_ctrl_t *const p_ctrl, uint32_t *data_size) |
| fsp_err_t(* | setupGet )(usb_ctrl_t *const p_ctrl, usb_setup_t *setup) |
| fsp_err_t(* | otgCallbackSet )(usb_ctrl_t *const p_ctrl, usb_otg_callback_t *p_callback) |
| fsp_err_t(* | otgSRP )(usb_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* usb_api_t::open) (usb_ctrl_t *const p_ctrl, usb_cfg_t const *const p_cfg)

Start the USB module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* usb_api_t::close) (usb_ctrl_t *const p_ctrl)

Stop the USB module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* usb_api_t::read) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

Request USB data read

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores read data. |
| [in] | size | Read request size. |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ write

fsp_err_t(* usb_api_t::write) (usb_ctrl_t *const p_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)

Request USB data write

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores write data. |
| [in] | size | Read request size. |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ stop

fsp_err_t(* usb_api_t::stop) (usb_ctrl_t *const p_ctrl, usb_transfer_t direction, uint8_t destination)

Stop USB data read/write processing

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | direction | Receive (USB_TRANSFER_READ) or send (USB_TRANSFER_WRITE). |
| [in] | destination | In Host mode, it represents the device address, and in Peripheral mode, it represents the device class. |

### ◆ suspend

fsp_err_t(* usb_api_t::suspend) (usb_ctrl_t *const p_ctrl)

Request suspend

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ resume

fsp_err_t(* usb_api_t::resume) (usb_ctrl_t *const p_ctrl)

Request resume

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ vbusSet

fsp_err_t(* usb_api_t::vbusSet) (usb_ctrl_t *const p_ctrl, uint16_t state)

Sets VBUS supply start/stop.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | state | VBUS supply start/stop specification |

### ◆ infoGet

fsp_err_t(* usb_api_t::infoGet) (usb_ctrl_t *const p_ctrl, usb_info_t *p_info, uint8_t destination)

Get information on USB device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to usb_info_t structure area. |
| [in] | destination | Device address for Host. |

### ◆ pipeRead

fsp_err_t(* usb_api_t::pipeRead) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data read from specified pipe

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores read data. |
| [in] | size | Read request size. |
| [in] | pipe_number | Pipe Number. |

### ◆ pipeWrite

fsp_err_t(* usb_api_t::pipeWrite) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data write to specified pipe

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buf | Pointer to area that stores write data. |
| [in] | size | Read request size. |
| [in] | pipe_number | Pipe Number. |

### ◆ pipeStop

fsp_err_t(* usb_api_t::pipeStop) (usb_ctrl_t *const p_ctrl, uint8_t pipe_number)

Stop USB data read/write processing to specified pipe

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pipe_number | Pipe Number. |

### ◆ usedPipesGet

fsp_err_t(* usb_api_t::usedPipesGet) (usb_ctrl_t *const p_ctrl, uint16_t *p_pipe, uint8_t destination)

Get pipe number

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_pipe | Pointer to area that stores the selected pipe number (bit map information). |
| [in] | destination | Device address for Host. |

### ◆ pipeInfoGet

fsp_err_t(* usb_api_t::pipeInfoGet) (usb_ctrl_t *const p_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)

Get pipe information

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to usb_pipe_t structure area. |
| [in] | pipe_number | Pipe Number. |

### ◆ eventGet

fsp_err_t(* usb_api_t::eventGet) (usb_ctrl_t *const p_ctrl, usb_status_t *event)

Return USB-related completed events (OS less only)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | event | Pointer to event. |

### ◆ callback

fsp_err_t(* usb_api_t::callback) (usb_callback_t *p_callback)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

**Parameters**

| | | |
|---|---|---|
| [in] | p_callback | Pointer to Callback function. |

### ◆ pullUp

fsp_err_t(* usb_api_t::pullUp) (usb_ctrl_t *const p_ctrl, uint8_t state)

Pull-up enable/disable setting of D+/D- line.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | state | Pull-up enable/disable setting. |

### ◆ hostControlTransfer

fsp_err_t(* usb_api_t::hostControlTransfer) (usb_ctrl_t *const p_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

Performs settings and transmission processing when transmitting a setup packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [in] | p_setup | Setup packet information. |
| [in] | p_buf | Transfer area information. |
| [in] | device_address | Device address information. |

### ◆ periControlDataGet

fsp_err_t(* usb_api_t::periControlDataGet) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)

Receives data sent by control transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [in] | p_buf | Data reception area information. |
| [in] | size | Data reception size information. |

### ◆ periControlDataSet

fsp_err_t(* usb_api_t::periControlDataSet) (usb_ctrl_t *const p_ctrl, uint8_t *p_buf, uint32_t size)

Performs transfer processing for control transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [in] | p_buf | Area information for data transfer. |
| [in] | size | Transfer size information. |

### ◆ periControlStatusSet

fsp_err_t(* usb_api_t::periControlStatusSet) (usb_ctrl_t *const p_ctrl, usb_setup_status_t status)

Set the response to the setup packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [in] | status | USB port startup information. |

### ◆ remoteWakeup

fsp_err_t(* usb_api_t::remoteWakeup) (usb_ctrl_t *const p_ctrl)

Sends a remote wake-up signal to the connected Host.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |

### ◆ driverActivate

fsp_err_t(* usb_api_t::driverActivate) (usb_ctrl_t *const p_api_ctrl)

Activate USB Driver

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |

### ◆ callbackMemorySet

fsp_err_t(* usb_api_t::callbackMemorySet) (usb_ctrl_t *const p_api_ctrl, usb_callback_args_t *p_callback_memory)

Set callback memory to USB driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | USB control structure. |
| [in] | p_callback_memory | Pointer to store USB event information. |

### ◆ moduleNumberGet

fsp_err_t(* usb_api_t::moduleNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *module_number)

This API gets the module number.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | module_number | Module number to get. |

### ◆ classTypeGet

fsp_err_t(* usb_api_t::classTypeGet) (usb_ctrl_t *const p_ctrl, usb_class_t *class_type)

This API gets the module number.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | class_type | Class type to get. |

### ◆ deviceAddressGet

fsp_err_t(* usb_api_t::deviceAddressGet) (usb_ctrl_t *const p_ctrl, uint8_t *device_address)

This API gets the device address.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | device_address | Device address to get. |

### ◆ pipeNumberGet

fsp_err_t(* usb_api_t::pipeNumberGet) (usb_ctrl_t *const p_ctrl, uint8_t *pipe_number)

This API gets the pipe number.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | pipe_number | Pipe number to get. |

### ◆ deviceStateGet

fsp_err_t(* usb_api_t::deviceStateGet) (usb_ctrl_t *const p_ctrl, uint16_t *state)

This API gets the state of the device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | state | Device state to get. |

### ◆ dataSizeGet

fsp_err_t(* usb_api_t::dataSizeGet) (usb_ctrl_t *const p_ctrl, uint32_t *data_size)

This API gets the data size.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | data_size | Data size to get. |

### ◆ setupGet

fsp_err_t(* usb_api_t::setupGet) (usb_ctrl_t *const p_ctrl, usb_setup_t *setup)

This API gets the setup type.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [out] | setup | Setup type to get. |

### ◆ otgCallbackSet

| fsp_err_t(* usb_api_t::otgCallbackSet) (usb_ctrl_t *const p_ctrl, usb_otg_callback_t *p_callback) |
|---|

This API sets the callback function for OTG.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |
| [in] | p_callback | Pointer to the callback function for OTG. |

### ◆ otgSRP

| fsp_err_t(* usb_api_t::otgSRP) (usb_ctrl_t *const p_ctrl) |
|---|

This API starts SRP processing for OTG.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | USB control structure. |

### ◆ usb_instance_t

| struct usb_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| usb_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| usb_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| usb_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ usb_ctrl_t

| typedef void usb_ctrl_t |
|---|
| USB control block. Allocate an instance specific control block to pass into the USB API calls. |

## Enumeration Type Documentation

### ◆ usb_speed_t

| enum usb_speed_t | |
|---|---|
| USB speed type | |
| Enumerator | |
| USB_SPEED_LS | Low speed operation. |
| USB_SPEED_FS | Full speed operation. |
| USB_SPEED_HS | Hi speed operation. |

### ◆ usb_setup_status_t

| enum usb_setup_status_t | |
|---|---|
| USB request result | |
| Enumerator | |
| USB_SETUP_STATUS_ACK | ACK response. |
| USB_SETUP_STATUS_STALL | STALL response. |

### ◆ usb_status_t

| enum usb_status_t | |
|---|---|
| USB driver status | |
| Enumerator | |
| USB_STATUS_POWERED | Powered State. |
| USB_STATUS_DEFAULT | Default State. |
| USB_STATUS_ADDRESS | Address State. |
| USB_STATUS_CONFIGURED | Configured State. |
| USB_STATUS_SUSPEND | Suspend State. |
| USB_STATUS_RESUME | Resume State. |
| USB_STATUS_DETACH | Detach State. |
| USB_STATUS_REQUEST | Request State. |
| USB_STATUS_REQUEST_COMPLETE | Request Complete State. |
| USB_STATUS_READ_COMPLETE | Read Complete State. |
| USB_STATUS_WRITE_COMPLETE | Write Complete State. |
| USB_STATUS_BC | battery Charge State |
| USB_STATUS_OVERCURRENT | Over Current state. |
| USB_STATUS_NOT_SUPPORT | Device Not Support. |
| USB_STATUS_NONE | None Status. |
| USB_STATUS_MSC_CMD_COMPLETE | MSC_CMD Complete. |

### ◆ usb_class_t

| enum usb_class_t | |
|---|---|
| USB class type | |
| Enumerator | |
| USB_CLASS_PCDC | PCDC Class. |

| | |
|---|---|
| USB_CLASS_PCDCC | PCDCC Class. |
| USB_CLASS_PCDC2 | PCDC2 Class. |
| USB_CLASS_PCDCC2 | PCDCC2 Class. |
| USB_CLASS_PHID | PHID Class. |
| USB_CLASS_PHID2 | PHID2 Class. |
| USB_CLASS_PAUD | PAUD Class. |
| USB_CLASS_PPRN | PPRN Class. |
| USB_CLASS_DFU | DFU Class. |
| USB_CLASS_PVND | PVND Class. |
| USB_CLASS_HCDC | HCDC Class. |
| USB_CLASS_HCDCC | HCDCC Class. |
| USB_CLASS_HHID | HHID Class. |
| USB_CLASS_HVND | HVND Class. |
| USB_CLASS_HMSC | HMSC Class. |
| USB_CLASS_PMSC | PMSC Class. |
| USB_CLASS_HPRN | HPRN Class. |
| USB_CLASS_HUVC | HUVC Class. |
| USB_CLASS_HAUD | HAUD Class. |
| USB_CLASS_REQUEST | USB Class Request. |
| USB_CLASS_HUB | HUB Class. |
| USB_CLASS_END | USB Class End Code. |

### ◆ usb_bcport_t

| enum usb_bcport_t | |
|---|---|
| USB battery charging type | |
| Enumerator | |
| USB_BCPORT_SDP | SDP port settings. |
| USB_BCPORT_CDP | CDP port settings. |
| USB_BCPORT_DCP | DCP port settings. |

### ◆ usb_onoff_t

| enum usb_onoff_t | |
|---|---|
| USB status | |
| Enumerator | |
| USB_OFF | USB Off State. |
| USB_ON | USB On State. |

### ◆ usb_transfer_t

| enum usb_transfer_t | |
|---|---|
| USB read/write type | |
| Enumerator | |
| USB_TRANSFER_READ | Data Receive communication. |
| USB_TRANSFER_WRITE | Data transmission communication. |

### ◆ usb_transfer_type_t

| enum usb_transfer_type_t | |
|---|---|
| USB transfer type | |
| Enumerator | |
| USB_TRANSFER_TYPE_BULK | Bulk communication. |
| USB_TRANSFER_TYPE_INT | Interrupt communication. |
| USB_TRANSFER_TYPE_ISO | Isochronous communication. |

### ◆ usb_mode_t

| enum usb_mode_t | |
|---|---|
| Enumerator | |
| USB_MODE_HOST | Host mode. |
| USB_MODE_PERI | Peripheral mode. |

### ◆ usb_compliancetest_status_t

| enum usb_compliancetest_status_t | |
|---|---|
| Enumerator | |
| USB_COMPLIANCETEST_ATTACH | Device Attach Detection. |
| USB_COMPLIANCETEST_DETACH | Device Detach Detection. |
| USB_COMPLIANCETEST_TPL | TPL device connect. |
| USB_COMPLIANCETEST_NOTTPL | Not TPL device connect. |
| USB_COMPLIANCETEST_HUB | USB Hub connect. |
| USB_COMPLIANCETEST_OVRC | Over current. |
| USB_COMPLIANCETEST_NORES | Response Time out for Control Read Transfer. |
| USB_COMPLIANCETEST_SETUP_ERR | Setup Transaction Error. |

### 5.3.5.17 USB PAUD Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB PAUD functions.

# Summary

The USB PAUD interface provides USB PAUD functionality.

#### Data Structures

| | |
|---|---|
| struct | usb_paud_audio_control_t |
| struct | usb_paud_audio_control_group_t |

#### Macros

| | |
|---|---|
| #define | USB_PAUD_CUR<br>CUR Request. |
| #define | USB_PAUD_RANGE<br>RANGE Request. |
| #define | USB_PAUD_SET_CUR<br>SET_CUR Request. |
| #define | USB_PAUD_GET_CUR<br>GET_CUR Request. |
| #define | USB_PAUD_GET_MIN<br>GET_MIN Request. |
| #define | USB_PAUD_GET_MAX<br>GET_MAX Request. |
| #define | USB_PAUD_GET_RES<br>GET_RES Request. |

| | |
|---|---|
| #define | USB_PAUD_MUTE_CONTROL<br>Mute Control in Feature Unit. |
| #define | USB_PAUD_VOLUME_CONTROL<br>Volume Control in Feature Unit. |
| #define | USB_PAUD_CLOCK_FREQUENCY_CONTROL<br>Clock Frequency Control. |

## Data Structure Documentation

### ◆ usb_paud_audio_control_t

| struct usb_paud_audio_control_t | | |
|---|---|---|
| Audio Control Settings | | |
| Data Fields | | |
| union usb_paud_audio_control_t | __unnamed__ | |
| uint32_t | sam_freq | Sampling Frequency. |
| uint32_t | fu_id | Feature Unit ID. |
| uint16_t | mute | Mute Control. |
| int16_t | volume_min | Minimum (Attribute of Volume Control) |
| int16_t | volume_max | Maximum (Attribute of Volume Control) |
| uint16_t | volume_res | Resolution (Attribute of Volume Control) |
| int16_t | volume | Volume Control. |

### ◆ usb_paud_audio_control_group_t

| struct usb_paud_audio_control_group_t | | |
|---|---|---|
| Audio Control Setting Group | | |
| Data Fields | | |
| uint32_t | controls_nb | Number of Audio Control Group. |
| usb_paud_audio_control_t * | p_controls | Pointer for access to Audio Control Settings. |

### 5.3.5.18 USB PCDC Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB PCDC functions.

# Summary

The USB PCDC interface provides USB PCDC functionality.

#### Data Structures

| | |
|---|---|
| struct | usb_serial_state_bitmap_t |
| union | usb_sci_serialstate_t |
| struct | usb_pcdc_linecoding_t |
| struct | usb_pcdc_ctrllinestate_t |

#### Macros

| | |
|---|---|
| #define | USB_PCDC_SET_LINE_CODING<br>Set Line Coding. |
| #define | USB_PCDC_GET_LINE_CODING<br>Get Line Coding. |
| #define | USB_PCDC_SET_CONTROL_LINE_STATE<br>Control Line State. |
| #define | USB_PCDC_SERIAL_STATE<br>Serial State Code. |
| #define | USB_PCDC_SETUP_TBL_BSIZE<br>Setup packet table size (uint16_t * 5) |

#### Data Structure Documentation

◆ **usb_serial_state_bitmap_t**

| struct usb_serial_state_bitmap_t | | |
|---|---|---|
| Virtual UART signal state | | |
| Data Fields | | |
| uint16_t | b_rx_carrier: 1 | DCD signal. |
| uint16_t | b_tx_carrier: 1 | DSR signal. |
| uint16_t | b_break: 1 | Break signal. |
| uint16_t | b_ring_signal: 1 | Ring signal. |
| uint16_t | b_framing: 1 | Framing error. |
| uint16_t | b_parity: 1 | Parity error. |
| uint16_t | b_over_run: 1 | Overrun error. |
| uint16_t | rsv: 9 | Reserved. |

### ◆ usb_sci_serialstate_t

| union usb_sci_serialstate_t | | |
|---|---|---|
| Class Notification Serial State | | |
| Data Fields | | |
| uint32_t | word | Word Access. |
| usb_serial_state_bitmap_t | bit | Bit Access. |

### ◆ usb_pcdc_linecoding_t

| struct usb_pcdc_linecoding_t | | |
|---|---|---|
| Virtual UART communication settings | | |
| Data Fields | | |
| uint32_t | dw_dte_rate | Bitrate. |
| uint8_t | b_char_format | Stop bits. |
| uint8_t | b_parity_type | Parity. |
| uint8_t | b_data_bits | Data bits. |
| uint8_t | rsv | Reserved. |

### ◆ usb_pcdc_ctrllinestate_t

| struct usb_pcdc_ctrllinestate_t | | |
|---|---|---|
| Virtual UART control line state | | |
| Data Fields | | |
| uint16_t | bdtr: 1 | DTR. |
| uint16_t | brts: 1 | RTS. |
| uint16_t | rsv: 14 | Reserved. |

### 5.3.5.19 USB PHID Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB PHID functions.

## Summary

The USB interface provides USB functionality.

### 5.3.5.20 USB PMSC Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB PMSC functions.

## Summary

The USB PMSC interface provides USB PMSC functionality.

#### Macros

| | |
|---|---|
| #define | USB_MASS_STORAGE_RESET<br>Mass storage reset request code. |
| #define | USB_GET_MAX_LUN<br>Get max logical unit number request code. |

### 5.3.5.21 USB PPRN Interface

Interfaces » Connectivity

#### Detailed Description

Interface for USB PPRN functions.

# Summary

The USB PPRN interface provides USB PPRN functionality.

### Macros

| | |
|---|---|
| #define | USB_PPRN_GET_DEVICE_ID<br>Get Device ID. |
| #define | USB_PRPN_GET_PORT_STATUS<br>Get Port Status. |
| #define | USB_PPRN_SOFT_RESET<br>Soft Reset. |
| #define | USB_PPRN_PORT_STATUS_PAPER_EMPTY<br>1: Paper Empty, 0: Paper Not Empty |
| #define | USB_PPRN_PORT_STATUS_SELECT<br>1: Selected, 0: Not Selected |
| #define | USB_PPRN_PORT_STATUS_NOT_ERROR<br>1: No Error, 0; Error |

### 5.3.5.22 USB TYPEC Interface

Interfaces » Connectivity

### Detailed Description

Interface for USB Type-C functions.

# Summary

The USB Type-C interface provides USB functionality.

### Data Structures

| | |
|---|---|
| struct | usb_typec_callback_args_t |

| | |
|---|---|
| struct | usb_typec_cfg_t |
| struct | usb_typec_api_t |
| struct | usb_typec_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | usb_typec_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | usb_typec_mode_t |
| enum | usb_typec_pmode_t |
| enum | usb_typec_plug_t |
| enum | usb_typec_connection_status_t |
| enum | usb_typec_vbus_status_t |
| enum | usb_typec_status_t |
| enum | usb_typec_ccs_status_t |

## Data Structure Documentation

### ◆ usb_typec_callback_args_t

| struct usb_typec_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | module_number | USB Type-C module number. |
| usb_typec_status_t | status | Status code. |
| usb_typec_plug_t | plug | Connection of Plug Orientation. |
| usb_typec_connection_status_t | connection_status | Status of Connection Statue Machine. |
| usb_typec_vbus_status_t | vbus_status | Status of VBUS. |
| void * | p_context | Placeholder for user data. Set in usb_typec_api_t::open function in usb_typec_cfg_t. |

### ◆ usb_typec_cfg_t

| struct usb_typec_cfg_t |
|---|
| |

USB Type-C configuration.

| Data Fields | | |
|---|---|---|
| uint32_t | module_number | USB Type-C module number. |
| usb_typec_mode_t | usb_typec_mode | Sink Only/USB 2.0 Only Sink Mode. |
| usb_typec_pmode_t | usb_typec_pmode | Detect Default USB Source/Detect Default USB and 1.5A Source/Detect Default USB, 1.5A and 3.0A Source. |
| IRQn_Type | irq_typec_cci | USBCC_CCI interrupt number storage variable. |
| IRQn_Type | irq_typec_ccs | USBCC_CCS interrupt number storage variable. |
| uint32_t | ipl_typec_cci | Variable to store the interrupt priority of USBCC_CCI IR. |
| uint32_t | ipl_typec_ccs | Variable to store the interrupt priority of USBCC_CCS IR. |
| usb_typec_callback_t * | p_callback | Type-C Callback. |
| void * | p_context | Other Context. |
| void const * | p_extend | Pointer to extended configuration by instance of interface. |

### ◆ usb_typec_api_t

struct usb_typec_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(usb_typec_ctrl_t *const p_ctrl, usb_typec_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(usb_typec_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(usb_typec_ctrl_t *const p_ctrl, usb_typec_info_t *const p_info) |
| | |
| fsp_err_t(* | lowPowerModeSet )(usb_typec_ctrl_t *const p_ctrl, usb_typec_ccs_status_t status) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* usb_typec_api_t::open) (usb_typec_ctrl_t *const p_ctrl, usb_typec_cfg_t const *const p_cfg)

Start the USB Type-C module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* usb_typec_api_t::close) (usb_typec_ctrl_t *const p_ctrl)

Stop the USB Type-C module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ infoGet

fsp_err_t(* usb_typec_api_t::infoGet) (usb_typec_ctrl_t *const p_ctrl, usb_typec_info_t *const p_info)

Get information on USB Type-C module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_info | Pointer to usb_typec_info_t structure area. |

### ◆ lowPowerModeSet

fsp_err_t(* usb_typec_api_t::lowPowerModeSet) (usb_typec_ctrl_t *const p_ctrl, usb_typec_ccs_status_t status)

Set IRQ Mode for USB Type-C module by Software Standby

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | status | Status code of USBCC CCS Enable/Disable. |

### ◆ usb_typec_instance_t

struct usb_typec_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| usb_typec_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| usb_typec_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| usb_typec_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ usb_typec_ctrl_t

typedef void usb_typec_ctrl_t

USB Type-C control block. Allocate an instance specific control block to pass into the USB Type-C API calls.

## Enumeration Type Documentation

### ◆ usb_typec_mode_t

enum usb_typec_mode_t

USB TypeC operation_mode

| Enumerator | |
|---|---|
| USB_TYPEC_MODE_SINK | Sink Only Mode. |
| USB_TYPEC_MODE_USB20_ONLY_SINK | USB 2.0 Only Sink Mode. |

### ◆ usb_typec_pmode_t

enum usb_typec_pmode_t

USB TypeC Source Current Detection mode

| Enumerator | |
|---|---|
| USB_TYPEC_PMODE_DEFAULT | Detect Default USB Source. |
| USB_TYPEC_PMODE_DEFAULT_1_5A | Detect Default USB and 1.5A Source. |
| USB_TYPEC_PMODE_DEFAULT_1_5A_3_0A | Detect Default USB, 1.5A and 3.0A Source. |

### ◆ usb_typec_plug_t

| enum usb_typec_plug_t | |
|---|---|
| USB TypeC Connection of Plug Orientation | |
| Enumerator | |
| USB_TYPEC_PLUG_CC1_CONNECTED | CC1 connected. |
| USB_TYPEC_PLUG_CC2_CONNECTED | CC2 connected. |

### ◆ usb_typec_connection_status_t

| enum usb_typec_connection_status_t | |
|---|---|
| USB TypeC Status of Connection State Machine | |
| Enumerator | |
| USB_TYPEC_CONNECTION_STATUS_DISABLED | Disabled. |
| USB_TYPEC_CONNECTION_STATUS_UNATTACHED | Unattached.SNK. |
| USB_TYPEC_CONNECTION_STATUS_ATTACHED_WAIT | AttachedWait.SNK. |
| USB_TYPEC_CONNECTION_STATUS_ATTACHED | Attached.SNK. |
| USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_DEFAULT | Attached.SNK (PowerDefault.SNK) |
| USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_1_5 | Attached.SNK (Power1.5.SNK) |
| USB_TYPEC_CONNECTION_STATUS_ATTACHED_POWER_3_0 | Attached.SNK (Power3.0.SNK) |

### ◆ usb_typec_vbus_status_t

| enum usb_typec_vbus_status_t | |
|---|---|
| USB TypeC VBUS status | |
| Enumerator | |
| USB_TYPEC_VBUS_STATUS_OFF | VBUS Off State. |
| USB_TYPEC_VBUS_STATUS_ON | VBUS On State. |

### ◆ usb_typec_status_t

| enum usb_typec_status_t | |
|---|---|
| Status code of callback function | |
| Enumerator | |
| USB_TYPEC_STATUS_ISCN | Connection Status Change. |
| USB_TYPEC_STATUS_ISCC | CC Status Change. |
| USB_TYPEC_STATUS_ISVBUS | VBUS Status Change. |
| USB_TYPEC_STATUS_ISVRA | Rp Detection for USBCC_CCI Interrupt. |
| USB_TYPEC_STATUS_ISVRA_CCS | Rp Detection for USBCC_CCS Interrupt. |

### ◆ usb_typec_ccs_status_t

| enum usb_typec_ccs_status_t | |
|---|---|
| Status code of USBCC CCS Enable/Disable | |
| Enumerator | |
| USB_TYPEC_CCS_STATUS_OFF | USBCC CCS Disable. |
| USB_TYPEC_CCS_STATUS_ON | USBCC CCS Enable. |

## 5.3.6 DSP

Interfaces

### Detailed Description

DSP Interfaces.

### Modules

IIR Interface

Interface for IIR filter functionality.

### 5.3.6.1 IIR Interface

Interfaces » DSP

#### Detailed Description

Interface for IIR filter functionality.

# Summary

The IIR interface allows access to the IIRFA peripheral for hardware acceleration of direct form 2 transposed biquad IIR filters.

#### Data Structures

| | |
|---|---|
| struct | iir_filter_coeffs_t |
| struct | iir_filter_state_t |
| struct | iir_filter_cfg_t |
| struct | iir_status_t |
| struct | iir_cfg_t |
| struct | iir_api_t |
| struct | iirfa_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | iir_ctrl_t |

#### Data Structure Documentation

◆ **iir_filter_coeffs_t**

| struct iir_filter_coeffs_t | | |
|---|---|---|
| Filter stage coefficient data | | |
| Data Fields | | |
| float | b0 | Coefficient B0. |
| float | b1 | Coefficient B1. |
| float | b2 | Coefficient B2. |
| float | a1 | Coefficient A0. |
| float | a2 | Coefficient A1. |

◆ **iir_filter_state_t**

| struct iir_filter_state_t | | |
|---|---|---|
| Filter stage state data | | |
| Data Fields | | |
| float | d0 | State variable D0. |
| float | d1 | State variable D1. |

### ◆ iir_filter_cfg_t

| struct iir_filter_cfg_t | | |
|---|---|---|
| Filter configuration | | |
| Data Fields | | |
| iir_filter_coeffs_t * | p_filter_coeffs | Filter coefficients. |
| iir_filter_state_t * | p_filter_state | Filter state data. |
| uint8_t | stage_base | Hardware stage to start from. |
| uint8_t | stage_num | Number of filter stages to use. |

### ◆ iir_status_t

| struct iir_status_t |
|---|
| Filter state register status |

### ◆ iir_cfg_t

| struct iir_cfg_t | | |
|---|---|---|
| IIRFA API configuration parameter | | |
| Data Fields | | |
| void const * | p_extend | |
| uint8_t | channel | IIRFA channel to use. |

### ◆ iir_api_t

| struct iir_api_t | |
|---|---|
| IIR driver structure. IIR functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(iir_ctrl_t *const p_ctrl, iir_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(iir_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | configure )(iir_ctrl_t *const p_ctrl, iir_filter_cfg_t const *const p_filter_cfg) |
| | |

| | |
|---|---|
| fsp_err_t(* | filter )(iir_ctrl_t *const p_ctrl, float const *p_data_in, float *p_data_out, uint16_t const num_samples) |
| | |
| fsp_err_t(* | statusGet )(iir_ctrl_t *const p_ctrl, iir_status_t *const p_status) |

## Field Documentation

### ◆ open

fsp_err_t(* iir_api_t::open) (iir_ctrl_t *const p_ctrl, iir_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* iir_api_t::close) (iir_ctrl_t *const p_ctrl)

Close the IIRFA channel.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in iir_api_t::open. |

### ◆ configure

fsp_err_t(* iir_api_t::configure) (iir_ctrl_t *const p_ctrl, iir_filter_cfg_t const *const p_filter_cfg)

Configure filter coefficients and state variables.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in iir_api_t::open. |
| [in] | p_filter_cfg | Pointer to filter configuration to write. |

### ◆ filter

| fsp_err_t(* iir_api_t::filter) (iir_ctrl_t *const p_ctrl, float const *p_data_in, float *p_data_out, uint16_t const num_samples) |
|---|

Filter the specified data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in iir_api_t::open. |
| [in] | p_data_in | Pointer to float input data. |
| [in] | p_data_out | Pointer to float output buffer. |
| [in] | num_samples | Number of samples to process. |

### ◆ statusGet

| fsp_err_t(* iir_api_t::statusGet) (iir_ctrl_t *const p_ctrl, iir_status_t *const p_status) |
|---|

Retrieve current status (including state registers).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in iir_api_t::open. |
| [in] | p_status | Pointer to status struct. |

### ◆ iirfa_instance_t

| struct iirfa_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| iir_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| iir_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| iir_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ iir_ctrl_t

| typedef void iir_ctrl_t |
|---|
| IIR control block. Allocate an instance specific control block to pass into the DAC API calls. |

## 5.3.7 Graphics

Interfaces

### Detailed Description

Graphics Interfaces.

### Modules

| | |
|---|---|
| CAPTURE Interface | Interface for CAPTURE functions. |
| Display Interface | Interface for LCD panel displays. |
| JPEG Codec Interface | Interface for JPEG functions. |
| MIPI CMD Data Types | Common data types for MIPI CSI and DSI communication interfaces. |
| MIPI DSI Interface | Interface for MIPI DSI communications. |
| SLCDC Interface | Interface for Segment LCD controllers. |

## 5.3.7.1 CAPTURE Interface

Interfaces » Graphics

### Detailed Description

Interface for CAPTURE functions.

# Summary

The CAPTURE interface provides the functionality for capturing an image from an image sensor/camera. When a capture is complete a capture complete interrupt is triggered.

### Data Structures

| | |
|---|---|
| struct | capture_status_t |
| struct | capture_callback_args_t |
| struct | capture_cfg_t |
| struct | capture_api_t |
| struct | capture_instance_t |

### Typedefs

| | |
|---|---|
| typedef uint32_t | capture_event_t |
| typedef void | capture_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | capture_state_t |

## Data Structure Documentation

### ◆ capture_status_t

| struct capture_status_t | | |
|---|---|---|
| CAPTURE status | | |
| Data Fields | | |
| capture_state_t | state | Current capture state. |
| uint32_t * | p_buffer | Pointer to active buffer. |
| uint32_t | data_size | Size of data written to provided buffer. |

### ◆ capture_callback_args_t

| struct capture_callback_args_t | | |
|---|---|---|
| CAPTURE callback function parameter data | | |
| Data Fields | | |
| capture_event_t | event | Event causing the callback - See instance header for details. |
| uint32_t | event_status | Capture status data |

| | | |
|---|---|---|
| | | corresponding to given event type - See instance header for details. |
| uint32_t | interrupt_status | Module interrupt status data corresponding to given event type - See instance header for details. |
| uint8_t * | p_buffer | Pointer to buffer that contains the most recently captured data. |
| void * | p_context | Placeholder for user data. Set in capture_api_t::open function in capture_cfg_t. |

### ◆ capture_cfg_t

| struct capture_cfg_t | |
|---|---|
| CAPTURE configuration parameters. | |
| **Data Fields** | |
| uint16_t | x_capture_start_pixel |
| | Horizontal position to start capture. |
| | |
| uint16_t | x_capture_pixels |
| | Number of horizontal pixels to capture. |
| | |
| uint16_t | y_capture_start_pixel |
| | Vertical position to start capture. |
| | |
| uint16_t | y_capture_pixels |
| | Number of vertical lines/pixels to capture. |
| | |
| uint8_t | bytes_per_pixel |
| | Number of bytes per pixel. |
| | |
| void(* | p_callback )(capture_callback_args_t *p_args) |
| | Callback provided when a CAPTURE transfer ISR occurs. |

| | |
|---|---|
| | |
| void * | p_context |
| | User defined context passed to callback function. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ capture_api_t

| struct capture_api_t | |
|---|---|
| CAPTURE functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(capture_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | captureStart )(capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer) |
| | |
| fsp_err_t(* | callbackSet )(capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | statusGet )(capture_ctrl_t *const p_ctrl, capture_status_t *p_status) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* capture_api_t::open) (capture_ctrl_t *const p_ctrl, capture_cfg_t const *const p_cfg)

Initial configuration.

***Note***

*To reconfigure after calling this function, call capture_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* capture_api_t::close) (capture_ctrl_t *const p_ctrl)

Closes the driver and allows reconfiguration. May reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ captureStart

fsp_err_t(* capture_api_t::captureStart) (capture_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

Start a capture.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buffer | New pointer to store captured image data. |

### ◆ callbackSet

fsp_err_t(* capture_api_t::callbackSet) (capture_ctrl_t *const p_ctrl, void(*p_callback)(capture_callback_args_t *), void *const p_context, capture_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CAPTURE control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ statusGet

fsp_err_t(* capture_api_t::statusGet) (capture_ctrl_t *const p_ctrl, capture_status_t *p_status)

Check scan status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_status | Pointer to store current status in |

### ◆ capture_instance_t

struct capture_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| capture_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| capture_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| capture_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ capture_event_t

| typedef uint32_t capture_event_t |
| --- |
| CAPTURE callback event ID - see implementation for details |

### ◆ capture_ctrl_t

| typedef void capture_ctrl_t |
| --- |
| CAPTURE control block. Allocate an instance specific control block to pass into the CAPTURE API calls. |

### Enumeration Type Documentation

### ◆ capture_state_t

| enum capture_state_t | |
| --- | --- |
| CAPTURE states. | |
| Enumerator | |
| CAPTURE_STATE_IDLE | CAPTURE is idle. |
| CAPTURE_STATE_IN_PROGRESS | CAPTURE capture in progress. |
| CAPTURE_STATE_BUSY | CAPTURE reset in progress. |

## 5.3.7.2 Display Interface

Interfaces » Graphics

### Detailed Description

Interface for LCD panel displays.

# Summary

The display interface provides standard display functionality:

- Signal timing configuration for LCD panels with RGB interface.
- Dot clock source selection (internal or external) and frequency divider.
- Blending of multiple graphics layers on the background screen.
- Color correction (brightness/configuration/gamma correction).
- Interrupts and callback function.

## Data Structures

| | |
|---|---|
| struct | display_timing_t |
| struct | display_color_t |
| struct | display_coordinate_t |
| struct | display_brightness_t |
| struct | display_contrast_t |
| struct | display_correction_t |
| struct | gamma_correction_t |
| struct | display_gamma_correction_t |
| struct | display_clut_t |
| struct | display_colorkeying_cfg_t |
| struct | display_colorkeying_layer_t |
| struct | display_input_cfg_t |
| struct | display_output_cfg_t |
| struct | display_layer_t |
| struct | display_callback_args_t |
| struct | display_cfg_t |
| struct | display_runtime_cfg_t |
| struct | display_clut_cfg_t |
| struct | display_status_t |
| struct | display_api_t |
| struct | display_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | display_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | display_frame_layer_t |
| enum | display_state_t |
| enum | display_event_t |
| enum | display_in_format_t |
| enum | display_out_format_t |
| enum | display_endian_t |
| enum | display_color_order_t |
| enum | display_signal_polarity_t |
| enum | display_sync_edge_t |
| enum | display_fade_control_t |
| enum | display_fade_status_t |
| enum | display_color_keying_t |
| enum | display_data_swap_t |

## Data Structure Documentation

### ◆ display_timing_t

| struct display_timing_t | | |
|---|---|---|
| Display signal timing setting | | |
| Data Fields | | |
| uint16_t | total_cyc | Total cycles in one line or total lines in one frame. |
| uint16_t | display_cyc | Active video cycles or lines. |
| uint16_t | back_porch | Back porch cycles or lines. |
| uint16_t | sync_width | Sync signal asserting width. |
| display_signal_polarity_t | sync_polarity | Sync signal polarity. |

### ◆ display_color_t

| struct display_color_t |
|---|
| RGB Color setting |

### ◆ display_coordinate_t

| struct display_coordinate_t | | |
|---|---|---|
| Contrast (gain) correction setting | | |
| Data Fields | | |
| int16_t | x | Coordinate X, this allows to set signed value. |
| int16_t | y | Coordinate Y, this allows to set signed value. |

### ◆ display_brightness_t

| struct display_brightness_t | | |
|---|---|---|
| Brightness (DC) correction setting | | |
| Data Fields | | |
| bool | enable | Brightness Correction On/Off. |
| uint16_t | r | Brightness (DC) adjustment for R channel. |
| uint16_t | g | Brightness (DC) adjustment for G channel. |
| uint16_t | b | Brightness (DC) adjustment for B channel. |

### ◆ display_contrast_t

| struct display_contrast_t | | |
|---|---|---|
| Contrast (gain) correction setting | | |
| Data Fields | | |
| bool | enable | Contrast Correction On/Off. |
| uint8_t | r | Contrast (gain) adjustment for R channel. |
| uint8_t | g | Contrast (gain) adjustment for G channel. |
| uint8_t | b | Contrast (gain) adjustment for B channel. |

### ◆ display_correction_t

| struct display_correction_t | | |
|---|---|---|
| Color correction setting | | |
| Data Fields | | |
| display_brightness_t | brightness | Brightness. |
| display_contrast_t | contrast | Contrast. |

### ◆ gamma_correction_t

| struct gamma_correction_t | | |
|---|---|---|
| Gamma correction setting for each color | | |
| Data Fields | | |
| bool | enable | Gamma Correction On/Off. |
| uint16_t * | gain | Gain adjustment. |
| uint16_t * | threshold | Start threshold. |

### ◆ display_gamma_correction_t

| struct display_gamma_correction_t | | |
|---|---|---|
| Gamma correction setting | | |
| Data Fields | | |
| gamma_correction_t | r | Gamma correction for R channel. |
| gamma_correction_t | g | Gamma correction for G channel. |
| gamma_correction_t | b | Gamma correction for B channel. |

### ◆ display_clut_t

| struct display_clut_t | | |
|---|---|---|
| CLUT setting | | |
| Data Fields | | |
| uint32_t | color_num | The number of colors in CLUT. |
| const uint32_t * | p_clut | Address of the area storing the CLUT data (in ARGB8888 format) |

### ◆ display_colorkeying_cfg_t

| struct display_colorkeying_cfg_t | | |
|---|---|---|
| Color Keying setting | | |
| Data Fields | | |
| display_color_t | src_color | Source color. |
| display_color_t | dst_color | Destination color. |
| display_color_keying_t | enable_ckey | Select enable or disable. |

### ◆ display_colorkeying_layer_t

| struct display_colorkeying_layer_t |
|---|
| Color Keying layer setting |

### ◆ display_input_cfg_t

| struct display_input_cfg_t | | |
|---|---|---|
| Graphics plane input configuration structure | | |
| Data Fields | | |
| uint32_t * | p_base | Base address to the frame buffer. |
| uint16_t | hsize | Horizontal pixel size in a line. |
| uint16_t | vsize | Vertical pixel size in a frame. |
| uint32_t | hstride | Memory stride (bytes) in a line. |
| display_in_format_t | format | Input format setting. |
| bool | line_descending_enable | Line descending enable. |
| bool | lines_repeat_enable | Line repeat enable. |
| uint16_t | lines_repeat_times | Expected number of line repeating. |

### ◆ display_output_cfg_t

| struct display_output_cfg_t | | |
|---|---|---|
| Display output configuration structure | | |
| Data Fields | | |
| display_timing_t | htiming | Horizontal display cycle setting. |
| display_timing_t | vtiming | Vertical display cycle setting. |
| display_out_format_t | format | Output format setting. |
| display_endian_t | endian | Bit order of output data. |
| display_color_order_t | color_order | Color order in pixel. |
| display_signal_polarity_t | data_enable_polarity | Data Enable signal polarity. |
| display_sync_edge_t | sync_edge | Signal sync edge selection. |
| display_color_t | bg_color | Background color. |
| display_brightness_t | brightness | Brightness setting. |
| display_contrast_t | contrast | Contrast setting. |
| display_gamma_correction_t * | p_gamma_correction | Pointer to gamma correction setting. |
| bool | dithering_on | Dithering on/off. |

### ◆ display_layer_t

| struct display_layer_t |
|---|
| Graphics layer blend setup parameter structure |

| Data Fields | | |
|---|---|---|
| display_coordinate_t | coordinate | Blending location (starting point of image) |
| display_color_t | bg_color | Color outside region. |
| display_fade_control_t | fade_control | Layer fade-in/out control on/off. |
| uint8_t | fade_speed | Layer fade-in/out frame rate. |

### ◆ display_callback_args_t

| struct display_callback_args_t | | |
|---|---|---|
| Display callback parameter definition | | |
| Data Fields | | |
| display_event_t | event | Event code. |
| void * | p_context | Context provided to user during callback. |

### ◆ display_cfg_t

| struct display_cfg_t | |
|---|---|
| Display main configuration structure | |
| **Data Fields** | |
| display_input_cfg_t | input [2] |
| | Graphics input frame setting. More... |
| | |
| display_output_cfg_t | output |
| | Graphics output frame setting. |
| | |
| display_layer_t | layer [2] |
| | Graphics layer blend setting. |
| | |
| uint8_t | line_detect_ipl |
| | Line detect interrupt priority. |
| | |
| uint8_t | underflow_1_ipl |
| | Underflow 1 interrupt priority. |

| | |
|---|---|
| | |
| uint8_t | underflow_2_ipl |
| | Underflow 2 interrupt priority. |
| | |
| IRQn_Type | line_detect_irq |
| | Line detect interrupt vector. |
| | |
| IRQn_Type | underflow_1_irq |
| | Underflow 1 interrupt vector. |
| | |
| IRQn_Type | underflow_2_irq |
| | Underflow 2 interrupt vector. |
| | |
| void(* | p_callback )(display_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | Display hardware dependent configuration. More... |
| | |

## Field Documentation

### ◆ input

display_input_cfg_t display_cfg_t::input[2]

Graphics input frame setting.

Generic configuration for display devices

### ◆ p_callback

| void(* display_cfg_t::p_callback) (display_callback_args_t *p_args) |
|---|
| Pointer to callback function.<br><br>Configuration for display event processing |

### ◆ p_extend

| void const* display_cfg_t::p_extend |
|---|
| Display hardware dependent configuration.<br><br>Pointer to display peripheral specific configuration |

### ◆ display_runtime_cfg_t

| struct display_runtime_cfg_t | | |
|---|---|---|
| Display main configuration structure | | |
| Data Fields | | |
| display_input_cfg_t | input | Graphics input frame setting.<br><br>Generic configuration for display devices |
| display_layer_t | layer | Graphics layer alpha blending setting. |

### ◆ display_clut_cfg_t

| struct display_clut_cfg_t | | |
|---|---|---|
| Display CLUT configuration structure | | |
| Data Fields | | |
| uint32_t * | p_base | Pointer to CLUT source data. |
| uint16_t | start | Beginning of CLUT entry to be updated. |
| uint16_t | size | Size of CLUT entry to be updated. |

### ◆ display_status_t

| struct display_status_t | | |
|---|---|---|
| Display Status | | |
| Data Fields | | |
| display_state_t | state | Status of display module. |
| display_fade_status_t | fade_status[ DISPLAY_FRAME_LAYER_2+1] | Status of fade-in/fade-out status. |

### ◆ display_api_t

| struct display_api_t | |
|---|---|
| Shared Interface definition for display peripheral | |
| **Data Fields** | |
| fsp_err_t(* | open )(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | start )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(display_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | layerChange )(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame) |
| | |
| fsp_err_t(* | bufferChange )(display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame) |
| | |
| fsp_err_t(* | correction )(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param) |
| | |
| fsp_err_t(* | clut )(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer) |
| | |
| fsp_err_t(* | clutEdit )(display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color) |
| | |
| fsp_err_t(* | colorKeySet )(display_ctrl_t const *const p_ctrl, display_colorkeying_layer_t key_cfg, display_frame_layer_t layer) |
| | |
| fsp_err_t(* | statusGet )(display_ctrl_t const *const p_ctrl, display_status_t *const p_status) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)

Open display device.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to display interface control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to display configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)

Close display device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ start

fsp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)

Display start.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ stop

fsp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)

Display stop.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ layerChange

fsp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)

Change layer parameters at runtime.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | p_cfg | Pointer to run-time layer configuration structure. |
| [in] | frame | Number of graphic frames. |

### ◆ bufferChange

fsp_err_t(* display_api_t::bufferChange) (display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)

Change layer framebuffer pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | framebuffer | Pointer to desired framebuffer. |
| [in] | frame | Number of graphic frames. |

### ◆ correction

fsp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)

Color correction.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | param | Pointer to color correction configuration structure. |

### ◆ clut

fsp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer)

Set CLUT for display device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | p_clut_cfg | Pointer to CLUT configuration structure. |
| [in] | layer | Layer number corresponding to the CLUT. |

### ◆ clutEdit

fsp_err_t(* display_api_t::clutEdit) (display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color)

Set CLUT element for display device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | layer | Layer number corresponding to the CLUT. |
| [in] | index | CLUT element index. |
| [in] | color | Desired CLUT index color. |

### ◆ colorKeySet

fsp_err_t(* display_api_t::colorKeySet) (display_ctrl_t const *const p_ctrl, display_colorkeying_layer_t key_cfg, display_frame_layer_t layer)

Configure color keying.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | key_cfg | Pointer to color keying configuration. |
| [in] | layer | Layer to apply color keying. |

### ◆ statusGet

| fsp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status) |
|---|

Get status for display device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | status | Pointer to display interface status structure. |

### ◆ display_instance_t

| struct display_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| display_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| display_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| display_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ display_ctrl_t

| typedef void display_ctrl_t |
|---|
| Display control block. Allocate an instance specific control block to pass into the display API calls.Display control block |

## Enumeration Type Documentation

### ◆ display_frame_layer_t

| enum display_frame_layer_t | |
|---|---|
| Display frame number | |
| Enumerator | |
| DISPLAY_FRAME_LAYER_1 | Frame layer 1. |
| DISPLAY_FRAME_LAYER_2 | Frame layer 2. |

### ◆ display_state_t

| enum display_state_t | |
|---|---|
| Display interface operation state | |
| Enumerator | |
| DISPLAY_STATE_CLOSED | Display closed. |
| DISPLAY_STATE_OPENED | Display opened. |
| DISPLAY_STATE_DISPLAYING | Displaying. |

### ◆ display_event_t

| enum display_event_t | |
|---|---|
| Display event codes | |
| Enumerator | |
| DISPLAY_EVENT_GR1_UNDERFLOW | Graphics frame1 underflow occurs. |
| DISPLAY_EVENT_GR2_UNDERFLOW | Graphics frame2 underflow occurs. |
| DISPLAY_EVENT_LINE_DETECTION | Designated line is processed. |
| DISPLAY_EVENT_FRAME_END | Frame end is processed. |

### ◆ display_in_format_t

| enum display_in_format_t | |
|---|---|
| Input format setting | |
| Enumerator | |
| DISPLAY_IN_FORMAT_32BITS_ARGB8888 | ARGB8888, 32 bits. |
| DISPLAY_IN_FORMAT_32BITS_RGB888 | RGB888, 32 bits. |
| DISPLAY_IN_FORMAT_16BITS_RGB565 | RGB565, 16 bits. |
| DISPLAY_IN_FORMAT_16BITS_ARGB1555 | ARGB1555, 16 bits. |
| DISPLAY_IN_FORMAT_16BITS_ARGB4444 | ARGB4444, 16 bits. |
| DISPLAY_IN_FORMAT_CLUT8 | CLUT8. |
| DISPLAY_IN_FORMAT_CLUT4 | CLUT4. |
| DISPLAY_IN_FORMAT_CLUT1 | CLUT1. |

### ◆ display_out_format_t

| enum display_out_format_t | |
|---|---|
| Output format setting | |
| Enumerator | |
| DISPLAY_OUT_FORMAT_24BITS_RGB888 | RGB888, 24 bits. |
| DISPLAY_OUT_FORMAT_18BITS_RGB666 | RGB666, 18 bits. |
| DISPLAY_OUT_FORMAT_16BITS_RGB565 | RGB565, 16 bits. |
| DISPLAY_OUT_FORMAT_8BITS_SERIAL | SERIAL, 8 bits. |

### ◆ display_endian_t

| enum display_endian_t | |
|---|---|
| Data endian select | |
| Enumerator | |
| DISPLAY_ENDIAN_LITTLE | Little-endian. |
| DISPLAY_ENDIAN_BIG | Big-endian. |

### ◆ display_color_order_t

| enum display_color_order_t | |
|---|---|
| RGB color order select | |
| Enumerator | |
| DISPLAY_COLOR_ORDER_RGB | Color order RGB. |
| DISPLAY_COLOR_ORDER_BGR | Color order BGR. |

### ◆ display_signal_polarity_t

| enum display_signal_polarity_t | |
|---|---|
| Polarity of a signal select | |
| Enumerator | |
| DISPLAY_SIGNAL_POLARITY_LOACTIVE | Low active signal. |
| DISPLAY_SIGNAL_POLARITY_HIACTIVE | High active signal. |

### ◆ display_sync_edge_t

| enum display_sync_edge_t | |
|---|---|
| Signal synchronization edge select | |
| Enumerator | |
| DISPLAY_SIGNAL_SYNC_EDGE_RISING | Signal is synchronized to rising edge. |
| DISPLAY_SIGNAL_SYNC_EDGE_FALLING | Signal is synchronized to falling edge. |

### ◆ display_fade_control_t

| enum display_fade_control_t | |
|---|---|
| Fading control | |
| Enumerator | |
| DISPLAY_FADE_CONTROL_NONE | Applying no fading control. |
| DISPLAY_FADE_CONTROL_FADEIN | Applying fade-in control. |
| DISPLAY_FADE_CONTROL_FADEOUT | Applying fade-out control. |

### ◆ display_fade_status_t

| enum display_fade_status_t | |
|---|---|
| Fading status | |
| Enumerator | |
| DISPLAY_FADE_STATUS_NOT_UNDERWAY | Fade-in/fade-out is not in progress. |
| DISPLAY_FADE_STATUS_FADING_UNDERWAY | Fade-in or fade-out is in progress. |
| DISPLAY_FADE_STATUS_PENDING | Fade-in/fade-out is configured but not yet started. |

### ◆ display_color_keying_t

| enum display_color_keying_t | |
|---|---|
| Color Keying enable or disable | |
| Enumerator | |
| DISPLAY_COLOR_KEYING_DISABLE | Color keying disable. |
| DISPLAY_COLOR_KEYING_ENABLE | Color keying enable. |

### ◆ display_data_swap_t

| enum display_data_swap_t |
|---|
| Data swap settings |

### 5.3.7.3 JPEG Codec Interface

Interfaces » Graphics

#### Detailed Description

Interface for JPEG functions.

#### Data Structures

| | |
|---|---|
| struct | jpeg_encode_image_size_t |
| struct | jpeg_callback_args_t |
| struct | jpeg_cfg_t |
| struct | jpeg_api_t |
| struct | jpeg_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | jpeg_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | jpeg_color_space_t |
| enum | jpeg_data_order_t |
| enum | jpeg_status_t |
| enum | jpeg_decode_pixel_format_t |
| enum | jpeg_decode_subsample_t |

#### Data Structure Documentation

◆ **jpeg_encode_image_size_t**

| struct jpeg_encode_image_size_t | | |
|---|---|---|
| Image parameter structure | | |
| Data Fields | | |
| uint16_t | horizontal_stride_pixels | Horizontal stride. |
| uint16_t | horizontal_resolution | Horizontal Resolution in pixels. |
| uint16_t | vertical_resolution | Vertical Resolution in pixels. |

### ◆ jpeg_callback_args_t

| struct jpeg_callback_args_t | | |
|---|---|---|
| Callback status structure | | |
| Data Fields | | |
| jpeg_status_t | status | JPEG status. |
| uint32_t | image_size | JPEG image size. |
| void * | p_context | Pointer to user-provided context. |

### ◆ jpeg_cfg_t

| struct jpeg_cfg_t | |
|---|---|
| User configuration structure, used in open function. | |
| **Data Fields** | |
| IRQn_Type | jedi_irq |
| | Data transfer interrupt IRQ number. |
| | |
| IRQn_Type | jdti_irq |
| | Decompression interrupt IRQ number. |
| | |
| uint8_t | jdti_ipl |
| | Data transfer interrupt priority. |
| | |
| uint8_t | jedi_ipl |
| | Decompression interrupt priority. |
| | |
| jpeg_mode_t | default_mode |
| | Mode to use at startup. |
| | |
| jpeg_data_order_t | decode_input_data_order |
| | Input data stream byte order. |
| | |
| | |

| | |
|---|---|
| jpeg_data_order_t | decode_output_data_order |
| | Output data stream byte order. |
| | |
| jpeg_decode_pixel_format_t | pixel_format |
| | Pixel format. |
| | |
| uint8_t | alpha_value |
| | Alpha value to be applied to decoded pixel data. Only valid for ARGB8888 format. |
| | |
| void(* | p_decode_callback )(jpeg_callback_args_t *p_args) |
| | User-supplied callback functions. |
| | |
| void * | p_decode_context |
| | Placeholder for user data. Passed to user callback in jpeg_callback_args_t. |
| | |
| jpeg_data_order_t | encode_input_data_order |
| | Input data stream byte order. |
| | |
| jpeg_data_order_t | encode_output_data_order |
| | Output data stream byte order. |
| | |
| uint16_t | dri_marker |
| | DRI Marker setting (0 = No DRI or RST marker) |
| | |
| uint16_t | horizontal_resolution |
| | Horizontal resolution of input image. |
| | |

| | |
|---|---|
| uint16_t | vertical_resolution |
| | Vertical resolution of input image. |
| | |
| uint16_t | horizontal_stride_pixels |
| | Horizontal stride of input image. |
| | |
| uint8_t const * | p_quant_luma_table |
| | Luma quantization table. |
| | |
| uint8_t const * | p_quant_chroma_table |
| | Chroma quantization table. |
| | |
| uint8_t const * | p_huffman_luma_ac_table |
| | Huffman AC table for luma. |
| | |
| uint8_t const * | p_huffman_luma_dc_table |
| | Huffman DC table for luma. |
| | |
| uint8_t const * | p_huffman_chroma_ac_table |
| | Huffman AC table for chroma. |
| | |
| uint8_t const * | p_huffman_chroma_dc_table |
| | Huffman DC table for chroma. |
| | |
| void(* | p_encode_callback )(jpeg_callback_args_t *p_args) |
| | User-supplied callback functions. |
| | |
| void * | p_encode_context |

| | |
|---|---|
| | Placeholder for user data. Passed to user callback in jpeg_callback_args_t. |
| | |

### ◆ jpeg_api_t

| struct jpeg_api_t | |
|---|---|
| JPEG functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | inputBufferSet )(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size) |
| | |
| fsp_err_t(* | outputBufferSet )(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size) |
| | |
| fsp_err_t(* | statusGet )(jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status) |
| | |
| fsp_err_t(* | close )(jpeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | horizontalStrideSet )(jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride) |
| | |
| fsp_err_t(* | pixelFormatGet )(jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space) |
| | |
| fsp_err_t(* | imageSubsampleSet )(jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample) |
| | |
| fsp_err_t(* | linesDecodedGet )(jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines) |
| | |
| fsp_err_t(* | imageSizeGet )(jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size) |
| | |
| fsp_err_t(* | imageSizeSet )(jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size) |

| | |
|---|---|
| fsp_err_t(* | modeSet )(jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode) |

## Field Documentation

### ◆ open

fsp_err_t(* jpeg_api_t::open) (jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg)

Initial configuration

**Precondition**
none

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ inputBufferSet

fsp_err_t(* jpeg_api_t::inputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign input data buffer to JPEG codec.

**Precondition**
the JPEG codec module must have been opened properly.

***Note***
*The buffer starting address must be 8-byte aligned.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | p_buffer | Pointer to the input buffer space |
| [in] | buffer_size | Size of the input buffer |

### ◆ outputBufferSet

fsp_err_t(* jpeg_api_t::outputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign output buffer to JPEG codec for storing output data.

**Precondition**

The JPEG codec module must have been opened properly.

***Note***

*The buffer starting address must be 8-byte aligned. For the decoding process, the HLD driver automatically computes the number of lines of the image to decoded so the output data fits into the given space. If the supplied output buffer is not able to hold the entire frame, the application should call the Output Full Callback function so it can be notified when additional buffer space is needed.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | p_buffer | Pointer to the output buffer space |
| [in] | buffer_size | Size of the output buffer |

### ◆ statusGet

fsp_err_t(* jpeg_api_t::statusGet) (jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status)

Retrieve current status of the JPEG codec module.

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_status | JPEG module status |

### ◆ close

fsp_err_t(* jpeg_api_t::close) (jpeg_ctrl_t *const p_ctrl)

Cancel an outstanding operation.

**Precondition**

the JPEG codec module must have been opened properly.

***Note***

*If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |

### ◆ horizontalStrideSet

fsp_err_t(* jpeg_api_t::horizontalStrideSet) (jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

Configure the horizontal stride value.

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | horizontal_stride | Horizontal stride value to be used for the decoded image data. |
| [in] | buffer_size | Size of the output buffer |

### ◆ pixelFormatGet

fsp_err_t(* jpeg_api_t::pixelFormatGet) (jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space)

Get the input pixel format.

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_color_space | JPEG input format. |

### ◆ imageSubsampleSet

fsp_err_t(* jpeg_api_t::imageSubsampleSet) (jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Configure the horizontal and vertical subsample settings.

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | horizontal_subsample | Horizontal subsample value |
| [in] | vertical_subsample | Vertical subsample value |

### ◆ linesDecodedGet

fsp_err_t(* jpeg_api_t::linesDecodedGet) (jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines)

Return the number of lines decoded into the output buffer.

**Precondition**

the JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_lines | Number of lines decoded |

### ◆ imageSizeGet

fsp_err_t(* jpeg_api_t::imageSizeGet) (jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Retrieve image size during decoding operation.

**Precondition**

the JPEG codec module must have been opened properly.

***Note***

*If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [out] | p_horizontal_size | Image horizontal size, in number of pixels. |
| [out] | p_vertical_size | Image vertical size, in number of pixels. |

### ◆ imageSizeSet

fsp_err_t(* jpeg_api_t::imageSizeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size)

Set image parameters to JPEG Codec

**Precondition**

The JPEG codec module must have been opened properly.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_image_size | Pointer to the RAW image parameters |

### ◆ modeSet

| fsp_err_t(* jpeg_api_t::modeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode) |
|---|

Switch between encode and decode mode or vice-versa.

**Precondition**

The JPEG codec module must have been opened properly. The JPEG Codec can only perform one operation at a time and requires different configuration for encode and decode. This function facilitates easy switching between the two modes in case both are needed in an application.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in jpeg_api_t::open call. |
| [in] | mode | Mode to switch to |

### ◆ jpeg_instance_t

| struct jpeg_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| jpeg_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| jpeg_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| jpeg_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ jpeg_ctrl_t

| typedef void jpeg_ctrl_t |
|---|
| JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls. |

## Enumeration Type Documentation

### ◆ jpeg_color_space_t

| enum jpeg_color_space_t | |
|---|---|
| Configuration for this module Image color space definitions | |
| Enumerator | |
| JPEG_COLOR_SPACE_YCBCR444 | Color Space YCbCr 444. |
| JPEG_COLOR_SPACE_YCBCR422 | Color Space YCbCr 422. |
| JPEG_COLOR_SPACE_YCBCR420 | Color Space YCbCr 420. |
| JPEG_COLOR_SPACE_YCBCR411 | Color Space YCbCr 411. |

### ◆ jpeg_data_order_t

| enum jpeg_data_order_t | |
|---|---|
| Multi-byte Data Format | |
| Enumerator | |
| JPEG_DATA_ORDER_NORMAL | (1)(2)(3)(4)(5)(6)(7)(8) Normal byte order |
| JPEG_DATA_ORDER_BYTE_SWAP | (2)(1)(4)(3)(6)(5)(8)(7) Byte Swap |
| JPEG_DATA_ORDER_WORD_SWAP | (3)(4)(1)(2)(7)(8)(5)(6) Word Swap |
| JPEG_DATA_ORDER_WORD_BYTE_SWAP | (4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap |
| JPEG_DATA_ORDER_LONGWORD_SWAP | (5)(6)(7)(8)(1)(2)(3)(4) Longword Swap |
| JPEG_DATA_ORDER_LONGWORD_BYTE_SWAP | (6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap |
| JPEG_DATA_ORDER_LONGWORD_WORD_SWAP | (7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap |
| JPEG_DATA_ORDER_LONGWORD_WORD_BYTE_SWAP | (8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap |

### ◆ jpeg_status_t

| enum jpeg_status_t | |
|---|---|
| JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status | |
| Enumerator | |
| JPEG_STATUS_NONE | JPEG codec module is not initialized. |
| JPEG_STATUS_IDLE | JPEG Codec module is open but not running. |
| JPEG_STATUS_RUNNING | JPEG Codec is running. |
| JPEG_STATUS_HEADER_PROCESSING | JPEG Codec module is reading the JPEG header information. |
| JPEG_STATUS_INPUT_PAUSE | JPEG Codec paused waiting for more input data. |
| JPEG_STATUS_OUTPUT_PAUSE | JPEG Codec paused after it decoded the number of lines specified by user. |
| JPEG_STATUS_IMAGE_SIZE_READY | JPEG decoding operation obtained image size, and paused. |
| JPEG_STATUS_ERROR | JPEG Codec module encountered an error. |
| JPEG_STATUS_OPERATION_COMPLETE | JPEG Codec has completed the operation. |

### ◆ jpeg_decode_pixel_format_t

| enum jpeg_decode_pixel_format_t | |
|---|---|
| Pixel Data Format | |
| Enumerator | |
| JPEG_DECODE_PIXEL_FORMAT_ARGB8888 | Pixel Data ARGB8888 format. |
| JPEG_DECODE_PIXEL_FORMAT_RGB565 | Pixel Data RGB565 format. |

### ◆ jpeg_decode_subsample_t

| enum jpeg_decode_subsample_t | |
|---|---|
| Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation. | |
| Enumerator | |
| JPEG_DECODE_OUTPUT_NO_SUBSAMPLE | No subsample. The image is decoded with no reduction in size. |
| JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF | The output image size is reduced by half. |
| JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER | The output image size is reduced to one-quarter. |
| JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH | The output image size is reduced to one-eighth. |

## 5.3.7.4 MIPI CMD Data Types

Interfaces » Graphics

### Detailed Description

Common data types for MIPI CSI and DSI communication interfaces.

# Summary

The MIPI CMD interface provides functionality involved with driving display panels over MIPI.

### Enumerations

| | |
|---|---|
| enum | mipi_cmd_id_t |

### Enumeration Type Documentation

### ◆ mipi_cmd_id_t

| enum mipi_cmd_id_t | |
|---|---|
| MIPI CMD packet Data Type (commands) - See MIPI specification for additional information | |
| Enumerator | |
| MIPI_CMD_ID_V_SYNC_START | (Short) Sync Event, V Sync Start |
| MIPI_CMD_ID_V_SYNC_END | (Short) Sync Event, V Sync End |
| MIPI_CMD_ID_H_SYNC_START | (Short) Sync Event, H Sync Start |
| MIPI_CMD_ID_H_SYNC_END | (Short) Sync Event, H Sync End |
| MIPI_CMD_ID_COMPRESSION_MODE | (Short) Compression Mode Command |
| MIPI_CMD_ID_END_OF_TRANSMISSION | (Short) End of Transmission packet (EoTp) |
| MIPI_CMD_ID_COLOR_MODE_OFF | (Short) Color Mode (CM) Off Command |
| MIPI_CMD_ID_COLOR_MODE_ON | (Short) Color Mode (CM) On Command |
| MIPI_CMD_ID_SHUTDOWN_PERIPHERAL | (Short) Shut Down Peripheral Command |
| MIPI_CMD_ID_TURN_ON_PERIPHERAL | (Short) Turn On Peripheral Command |
| MIPI_CMD_ID_GENERIC_SHORT_WRITE_0_PARAM | (Short) Generic Short WRITE, no parameters |
| MIPI_CMD_ID_GENERIC_SHORT_WRITE_1_PARAM | (Short) Generic Short WRITE, 1 parameter |
| MIPI_CMD_ID_GENERIC_SHORT_WRITE_2_PARAM | (Short) Generic Short WRITE, 2 parameters |
| MIPI_CMD_ID_GENERIC_READ_REQUEST_0_PARAM | (Short) Generic READ, no parameters |
| MIPI_CMD_ID_GENERIC_READ_REQUEST_1_PARAM | (Short) Generic READ, 1 parameter |
| MIPI_CMD_ID_GENERIC_READ_REQUEST_2_PARAM | (Short) Generic READ, 2 parameters |
| MIPI_CMD_ID_DCS_SHORT_WRITE_0_PARAM | (Short) DCS Short WRITE, no parameters |
| MIPI_CMD_ID_DCS_SHORT_WRITE_1_PARAM | (Short) DCS Short WRITE, 1 parameter |
| MIPI_CMD_ID_DCS_READ | (Short) DCS READ, no parameters |
| MIPI_CMD_ID_EXECUTE_QUEUE | (Short) Execute Queue |
| MIPI_CMD_ID_SET_MAXIMUM_RETURN_PACKET_S | (Short) Set Maximum Return Packet Size |

| | |
|---|---|
| IZE | |
| MIPI_CMD_ID_NULL_PACKET | (Long) Null Packet, no data |
| MIPI_CMD_ID_BLANKING_PACKET | (Long) Blanking Packet, no data |
| MIPI_CMD_ID_GENERIC_LONG_WRITE | (Long) Generic Long Write |
| MIPI_CMD_ID_DCS_LONG_WRITE | (Long) DCS Long Write/write_LUT Command Packet |
| MIPI_CMD_ID_PICTURE_PARAMETER_SET | (Long) Picture Parameter Set |
| MIPI_CMD_ID_COMPRESSED_PIXEL_STREAM | (Long) Compressed Pixel Stream |
| MIPI_CMD_ID_LOOSELY_PACKED_PIXEL_STREAM_YCBCR20 | (Long) Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR24 | (Long) Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR16 | (Long) Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_30 | (Long) Packed Pixel Stream, 30-bit RGB, 10-10-10 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_36 | (Long) Packed Pixel Stream, 36-bit RGB, 12-12-12 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR12 | (Long) Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_16 | (Long) Packed Pixel Stream, 16-bit RGB, 5-6-5 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_18 | (Long) Packed Pixel Stream, 18-bit RGB, 6-6-6 Format |
| MIPI_CMD_ID_LOOSELY_PACKED_PIXEL_STREAM_18 | (Long) Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format |
| MIPI_CMD_ID_PACKED_PIXEL_STREAM_24 | (Long) Packed Pixel Stream, 24-bit RGB, 8-8-8 Format |

### 5.3.7.5 MIPI DSI Interface

Interfaces » Graphics

### Detailed Description

Interface for MIPI DSI communications.

# Summary

The MIPI DSI interface provides functionality involved with driving display panels over MIPI.

### Data Structures

| | |
|---|---|
| struct | mipi_dsi_cmd_t |
| union | mipi_dsi_ack_err_status_t |
| struct | mipi_dsi_status_t |
| struct | mipi_dsi_callback_args_t |
| struct | mipi_dsi_timing_t |
| struct | mipi_dsi_cfg_t |
| struct | mipi_dsi_api_t |
| struct | mipi_dsi_instance_t |

### Typedefs

| | |
|---|---|
| typedef __PACKED_STRUCT | st_mipi_dsi_result<br>Data of received packet header. More... |
| typedef void | mipi_dsi_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | mipi_dsi_cmd_id_t |
| enum | mipi_dsi_dcs_id_t |
| enum | mipi_dsi_video_data_t |
| enum | mipi_dsi_ack_err_t |
| enum | mipi_dsi_vc_t |
| enum | mipi_dsi_cmd_flag_t |

| | |
|---|---|
| enum | mipi_dsi_event_t |
| enum | mipi_dsi_sequence_status_t |
| enum | mipi_dsi_video_status_t |
| enum | mipi_dsi_receive_status_t |
| enum | mipi_dsi_fatal_status_t |
| enum | mipi_dsi_phy_status_t |
| enum | mipi_dsi_link_status_t |
| enum | mipi_dsi_lane_t |

## Variables

| | |
|---|---|
| mipi_cmd_id_t | cmd_id<br>Data type. |
| uint8_t | virtual_channel_id<br>Virtual channel ID. |
| uint8_t | long_packet<br>Sort packet (0) or Long packet (1) |
| uint8_t | rx_success<br>Response packet or ack trigger received. |
| uint8_t | timeout<br>Fatal timeout error. |
| uint8_t | rx_fail<br>Expected receive not done. |
| uint8_t | rx_data_fail<br>Receive packet data fail. |

| | |
|---|---|
| uint8_t | rx_correctable_error<br>Correctable error detected. |
| uint8_t | rx_ack_err<br>Rx acknowledge and error report packet received. |
| uint8_t | info_overwrite<br>This information was overwritten. |

## Data Structure Documentation

### ◆ mipi_dsi_cmd_t

| struct mipi_dsi_cmd_t | | |
|---|---|---|
| MIPI DSI Command | | |
| Data Fields | | |
| uint8_t | channel | Virtual Channel ID. |
| mipi_cmd_id_t | cmd_id | Message ID. |
| mipi_dsi_cmd_flag_t | flags | Flags controlling this message transition. |
| uint16_t | tx_len | Transmit buffer size. |
| const uint8_t * | p_tx_buffer | Transmit buffer pointer. |
| const uint8_t * | p_rx_buffer | Receive buffer pointer. |

### ◆ mipi_dsi_ack_err_status_t

| union mipi_dsi_ack_err_status_t | | |
|---|---|---|
| MIPI DSI Acknowledge and Error status type | | |
| Data Fields | | |
| | __PACKED_STRUCT | Error report bits. |
| mipi_dsi_vc_t | virtual_channel: 4 | Virtual Channel ID. |
| uint32_t | __pad0__: 12 | |
| uint32_t | bits | |

### ◆ mipi_dsi_status_t

| struct mipi_dsi_status_t |
|---|
| MIPI DSI status type |
| Data Fields |

| | | |
|---|---|---|
| mipi_dsi_link_status_t | link_status | Link status. |
| mipi_dsi_ack_err_status_t | ack_err_latest | Latest Acknowledge and Error Report Packet Latest Info. |
| mipi_dsi_ack_err_status_t | ack_err_accumulated | Accumulated Acknowledge and Error Report Packet Latest Info. |

### ◆ mipi_dsi_callback_args_t

| struct mipi_dsi_callback_args_t | | |
|---|---|---|
| MIPI DSI callback parameter definition | | |
| Data Fields | | |
| mipi_dsi_event_t | event | Event code. |
| union mipi_dsi_callback_args_t | __unnamed__ | |
| mipi_dsi_receive_result_t * | p_result | Receive result pointer. |
| void * | p_context | Context provided to user during callback. |

### ◆ mipi_dsi_timing_t

| struct mipi_dsi_timing_t | | |
|---|---|---|
| MIPI DSI transition timing | | |
| Data Fields | | |
| uint32_t | clock_stop_time | Clock stop time. |
| uint32_t | clock_beforehand_time | Clock beforehand time. |
| uint32_t | clock_keep_time | Clock Keep time. |
| uint32_t | go_lp_and_back | Go LP and Back time. |

### ◆ mipi_dsi_cfg_t

| struct mipi_dsi_cfg_t | |
|---|---|
| MIPI DSI main configuration structure | |
| **Data Fields** | |
| mipi_phy_instance_t const * | p_mipi_phy_instance |
| | Pointer to mipi physical layer instance. |
| | |
| mipi_dsi_timing_t const * | p_timing |
| | Pointer to MIPI DSI timing configuration. |
| | |
| bool | hsa_no_lp |

| | |
|---|---|
| | Suppress the transition to LP during HSA period and keep HS. |
| | |
| bool | hbp_no_lp |
| | Suppress the transition to LP during HBP period and keep HS. |
| | |
| bool | hfp_no_lp |
| | Suppress the transition to LP during HFP period and keep HS. |
| | |
| uint8_t | num_lanes |
| | Number of MIPI lanes to use. |
| | |
| uint8_t | ulps_wakeup_period |
| | ULPS wakeup period. |
| | |
| uint8_t | continuous_clock |
| | Always run HS clock on/off. |
| | |
| uint32_t | hs_tx_timeout |
| | HS-Tx Timeout value. |
| | |
| uint32_t | lp_rx_timeout |
| | LP-Rx host processor timeout. |
| | |
| uint32_t | turnaround_timeout |
| | Turnaround Acknowledge Timeout. |
| | |
| uint32_t | bta_timeout |
| | Peripheral Response Timeout. |

| | |
|---|---|
| | |
| uint32_t | lprw_timeout |
| | LP Read and Write Timeouts. |
| | |
| uint32_t | hsrw_timeout |
| | HS Read and Write Timeouts. |
| | |
| uint32_t | max_return_packet_size |
| | Maximum return packet size. |
| | |
| bool | ecc_enable |
| | ECC Check enable. |
| | |
| mipi_dsi_vc_t | crc_check_mask |
| | Virtual channel CRC check enable. |
| | |
| bool | scramble_enable |
| | Scramble on/off. |
| | |
| bool | tearing_detect |
| | External tearing effect detection mode (0:rising, 1:falling edge) |
| | |
| bool | eotp_enable |
| | End of Transmit Packet (EoTP) on/off. |
| | |
| bool | sync_pulse |
| | Enable for Non-Burst Mode with Sync Pulse sequence. |
| | |

| | |
|---|---|
| mipi_dsi_video_data_t | data_type |
| | Video mode data type: 16-bit RGB, 18-bit RGB, 24-bit RGB. |
| | |
| uint8_t | virtual_channel_id |
| | Video mode virtual channel to use (from 0x0 to 0x3) |
| | |
| uint32_t | vertical_sync_lines |
| | Number of vertical sync active lines. |
| | |
| bool | vertical_sync_polarity |
| | V-Sync Polarity. |
| | |
| uint32_t | vertical_active_lines |
| | Number of vertical active lines. |
| | |
| uint32_t | vertical_back_porch |
| | Vertical back porch. |
| | |
| uint32_t | vertical_front_porch |
| | Vertical front porch. |
| | |
| uint32_t | horizontal_sync_lines |
| | Number of horizontal sync active lines. |
| | |
| bool | horizontal_sync_polarity |
| | H-Sync Polarity. |
| | |
| uint32_t | horizontal_active_lines |

| | |
|---|---|
| | Number of horizontal active lines. |
| | |
| uint32_t | horizontal_back_porch |
| | Horizontal back porch. |
| | |
| uint32_t | horizontal_front_porch |
| | Horizontal front porch. |
| | |
| void(* | p_callback )(mipi_dsi_callback_args_t *p_args) |
| | Pointer to callback function. More... |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | MIPI hardware dependent configuration. More... |
| | |

## Field Documentation

### ◆ p_callback

| |
|---|
| void(* mipi_dsi_cfg_t::p_callback) (mipi_dsi_callback_args_t *p_args) |
| Pointer to callback function.<br>Callback configuration |

### ◆ p_extend

| |
|---|
| void const* mipi_dsi_cfg_t::p_extend |
| MIPI hardware dependent configuration.<br>Pointer to display peripheral specific configuration |

### ◆ mipi_dsi_api_t

| |
|---|
| struct mipi_dsi_api_t |
| Shared Interface definition for MIPI DSI peripheral |

| Data Fields | |
|---|---|
| fsp_err_t(* | open )(mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(mipi_dsi_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | start )(mipi_dsi_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(mipi_dsi_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | ulpsEnter )(mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_lane_t lane) |
| | |
| fsp_err_t(* | ulpsExit )(mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_lane_t lane) |
| | |
| fsp_err_t(* | command )(mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_cmd_t *p_cmd) |
| | |
| fsp_err_t(* | statusGet )(mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_status_t *p_status) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* mipi_dsi_api_t::open) (mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_cfg_t const *const p_cfg)

Open MIPI DSI device.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to MIPI DSI interface control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to MIPI DSI configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* mipi_dsi_api_t::close) (mipi_dsi_ctrl_t *const p_ctrl)

Close MIPI DSI device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |

### ◆ start

fsp_err_t(* mipi_dsi_api_t::start) (mipi_dsi_ctrl_t *const p_ctrl)

Start pixel data output.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |

### ◆ stop

fsp_err_t(* mipi_dsi_api_t::stop) (mipi_dsi_ctrl_t *const p_ctrl)

Stop pixel data output.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |

### ◆ ulpsEnter

fsp_err_t(* mipi_dsi_api_t::ulpsEnter) (mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_lane_t lane)

Enter Ultra-low Power State (ULPS).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |
| [in] | lane | Physical lane(s) to transition into ULPS |

### ◆ ulpsExit

fsp_err_t(* mipi_dsi_api_t::ulpsExit) (mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_lane_t lane)

Exit Ultra-low Power State (ULPS).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |
| [in] | lane | Physical lane(s) to transition from ULPS |

### ◆ command

fsp_err_t(* mipi_dsi_api_t::command) (mipi_dsi_ctrl_t *const p_api_ctrl, mipi_dsi_cmd_t *p_cmd)

Send a command to the display.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |
| [in] | p_cmd | Pointer to a command structure |

### ◆ statusGet

fsp_err_t(* mipi_dsi_api_t::statusGet) (mipi_dsi_ctrl_t *const p_ctrl, mipi_dsi_status_t *p_status)

Get status of MIPI link.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to MIPI DSI interface control block. |
| [in] | p_status | Pointer to MIPI DSI interface status structure. |

### ◆ mipi_dsi_instance_t

struct mipi_dsi_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| mipi_dsi_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| mipi_dsi_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| mipi_dsi_api_t const * | p_api | Pointer to the API structure for |

| | | |
|---|---|---|
| | | this instance. |

## Typedef Documentation

### ◆ st_mipi_dsi_result

| typedef __PACKED_STRUCT st_mipi_dsi_result |
|---|
| Data of received packet header.<br>MIPI DSI Result |

### ◆ mipi_dsi_ctrl_t

| typedef void mipi_dsi_ctrl_t |
|---|
| MIPI DSI control block. Allocate an instance specific control block to pass into the MIPI DSI API calls. |

## Enumeration Type Documentation

### ◆ mipi_dsi_cmd_id_t

enum mipi_dsi_cmd_id_t

DEPRECATED - MIPI DSI packet Data Type (commands) - See MIPI specification for additional information

- The use of mipi_dsi_cmd_id_t has been deprecated and will be fully relpaced with the use of mipi_cmd_t in a future release.

| Enumerator | |
|---|---|
| MIPI_DSI_CMD_ID_V_SYNC_START | (Short) Sync Event, V Sync Start |
| MIPI_DSI_CMD_ID_V_SYNC_END | (Short) Sync Event, V Sync End |
| MIPI_DSI_CMD_ID_H_SYNC_START | (Short) Sync Event, H Sync Start |
| MIPI_DSI_CMD_ID_H_SYNC_END | (Short) Sync Event, H Sync End |
| MIPI_DSI_CMD_ID_COMPRESSION_MODE | (Short) Compression Mode Command |
| MIPI_DSI_CMD_ID_END_OF_TRANSMISSION | (Short) End of Transmission packet (EoTp) |
| MIPI_DSI_CMD_ID_COLOR_MODE_OFF | (Short) Color Mode (CM) Off Command |
| MIPI_DSI_CMD_ID_COLOR_MODE_ON | (Short) Color Mode (CM) On Command |
| MIPI_DSI_CMD_ID_SHUTDOWN_PERIPHERAL | (Short) Shut Down Peripheral Command |
| MIPI_DSI_CMD_ID_TURN_ON_PERIPHERAL | (Short) Turn On Peripheral Command |
| MIPI_DSI_CMD_ID_GENERIC_SHORT_WRITE_0_PARAM | (Short) Generic Short WRITE, no parameters |
| MIPI_DSI_CMD_ID_GENERIC_SHORT_WRITE_1_PARAM | (Short) Generic Short WRITE, 1 parameter |
| MIPI_DSI_CMD_ID_GENERIC_SHORT_WRITE_2_PARAM | (Short) Generic Short WRITE, 2 parameters |
| MIPI_DSI_CMD_ID_GENERIC_READ_REQUEST_0_PARAM | (Short) Generic READ, no parameters |
| MIPI_DSI_CMD_ID_GENERIC_READ_REQUEST_1_PARAM | (Short) Generic READ, 1 parameter |
| MIPI_DSI_CMD_ID_GENERIC_READ_REQUEST_2_PARAM | (Short) Generic READ, 2 parameters |
| MIPI_DSI_CMD_ID_DCS_SHORT_WRITE_0_PARAM | (Short) DCS Short WRITE, no parameters |
| MIPI_DSI_CMD_ID_DCS_SHORT_WRITE_1_PARAM | (Short) DCS Short WRITE, 1 parameter |
| MIPI_DSI_CMD_ID_DCS_READ | |

| | |
|---|---|
| | (Short) DCS READ, no parameters |
| MIPI_DSI_CMD_ID_EXECUTE_QUEUE | (Short) Execute Queue |
| MIPI_DSI_CMD_ID_SET_MAXIMUM_RETURN_PACKET_SIZE | (Short) Set Maximum Return Packet Size |
| MIPI_DSI_CMD_ID_NULL_PACKET | (Long) Null Packet, no data |
| MIPI_DSI_CMD_ID_BLANKING_PACKET | (Long) Blanking Packet, no data |
| MIPI_DSI_CMD_ID_GENERIC_LONG_WRITE | (Long) Generic Long Write |
| MIPI_DSI_CMD_ID_DCS_LONG_WRITE | (Long) DCS Long Write/write_LUT Command Packet |
| MIPI_DSI_CMD_ID_PICTURE_PARAMETER_SET | (Long) Picture Parameter Set |
| MIPI_DSI_CMD_ID_COMPRESSED_PIXEL_STREAM | (Long) Compressed Pixel Stream |
| MIPI_DSI_CMD_ID_LOOSELY_PACKED_PIXEL_STREAM_YCBCR20 | (Long) Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR24 | (Long) Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR16 | (Long) Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_30 | (Long) Packed Pixel Stream, 30-bit RGB, 10-10-10 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_36 | (Long) Packed Pixel Stream, 36-bit RGB, 12-12-12 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_YCBCR12 | (Long) Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_16 | (Long) Packed Pixel Stream, 16-bit RGB, 5-6-5 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_18 | (Long) Packed Pixel Stream, 18-bit RGB, 6-6-6 Format |
| MIPI_DSI_CMD_ID_LOOSELY_PACKED_PIXEL_STREAM_18 | (Long) Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format |
| MIPI_DSI_CMD_ID_PACKED_PIXEL_STREAM_24 | (Long) Packed Pixel Stream, 24-bit RGB, 8-8-8 Format |

### ◆ mipi_dsi_dcs_id_t

| enum mipi_dsi_dcs_id_t | |
|---|---|
| MIPI DCS ID types - See MIPI DCS specification for additional information | |
| Enumerator | |
| MIPI_DSI_DCS_ID_ENTER_IDLE_MODE | Enter idle mode. |
| MIPI_DSI_DCS_ID_ENTER_INVERT_MODE | Displayed image colors inverted. |
| MIPI_DSI_DCS_ID_ENTER_NORMAL_MODE | Whole display area used for image. |
| MIPI_DSI_DCS_ID_ENTER_PARTIAL_MODE | Part of display area used for image. |
| MIPI_DSI_DCS_ID_ENTER_SLEEP_MODE | Power off the display panel. |
| MIPI_DSI_DCS_ID_EXIT_IDLE_MODE | Full color depth used. |
| MIPI_DSI_DCS_ID_EXIT_INVERT_MODE | Displayed image colors not inverted. |
| MIPI_DSI_DCS_ID_EXIT_SLEEP_MODE | Power on the display panel. |
| MIPI_DSI_DCS_ID_GET_3D_CONTROL | Get display module 3D mode. |
| MIPI_DSI_DCS_ID_GET_ADDRESS_MODE | Get data order for transfers from host to the display device. |
| MIPI_DSI_DCS_ID_GET_BLUE_CHANNEL | Get blue component of pixel at 0,0. |
| MIPI_DSI_DCS_ID_GET_CABC_MIN_BRIGHTNESS | Get current minimum brightness level of the active CABC mode. |
| MIPI_DSI_DCS_ID_GET_COMPRESSION_MODE | Get current compression mode. |
| MIPI_DSI_DCS_ID_GET_CONTROL_DISPLAY | Get control display mode. |
| MIPI_DSI_DCS_ID_GET_DIAGNOSTIC_RESULT | Get peripheral self-diagnostic result. |
| MIPI_DSI_DCS_ID_GET_DISPLAY_BRIGHTNESS | Get current display brightness level. |
| MIPI_DSI_DCS_ID_GET_DISPLAY_MODE | Get current display mode from the peripheral. |
| MIPI_DSI_DCS_ID_GET_DSI_MODE | Get DSI operation mode. |
| MIPI_DSI_DCS_ID_GET_ERROR_COUNT_ON_DSI | Get number of corrupted packets on DSI. |
| MIPI_DSI_DCS_ID_GET_GREEN_CHANNEL | Get green component of pixel at 0,0. |

| | |
|---|---|
| MIPI_DSI_DCS_ID_GET_IMAGE_CHECKSUM_CT | Returns checksum of frame of color-transformed pixel data. |
| MIPI_DSI_DCS_ID_GET_IMAGE_CHECKSUM_RGB | Returns checksum of frame of RGB pixel data. |
| MIPI_DSI_DCS_ID_GET_PIXEL_FORMAT | Get current pixel format. |
| MIPI_DSI_DCS_ID_GET_POWER_MODE | Get current power mode. |
| MIPI_DSI_DCS_ID_GET_POWER_SAVE | Get current power-save mode. |
| MIPI_DSI_DCS_ID_GET_RED_CHANNEL | Get red component of pixel at 0,0. |
| MIPI_DSI_DCS_ID_GET_SCANLINE | Get current scanline. |
| MIPI_DSI_DCS_ID_GET_SIGNAL_MODE | Get display module signaling mode. |
| MIPI_DSI_DCS_ID_NOP | No operation. |
| MIPI_DSI_DCS_ID_READ_ACMD | Perform read access to the ACMD registers. |
| MIPI_DSI_DCS_ID_READ_DDB_CONTINUE | Continue reading the DDB from the last read location. |
| MIPI_DSI_DCS_ID_READ_DDB_START | Read the DDB from the provided location. |
| MIPI_DSI_DCS_ID_READ_DSE_MAILBOX | Read access to the registers of the DSE read or write control mailbox. |
| MIPI_DSI_DCS_ID_READ_MEMORY_CONTINUE | Read image data from peripheral, continuing after last read. |
| MIPI_DSI_DCS_ID_READ_MEMORY_START | Read image data from the peripheral to the host. |
| MIPI_DSI_DCS_ID_READ_PPS_CONTINUE | Continue reading the specified length of PPS data. |
| MIPI_DSI_DCS_ID_READ_PPS_START | Read PPS data. |
| MIPI_DSI_DCS_ID_SET_3D_CONTROL | 3D is used on the display panel |
| MIPI_DSI_DCS_ID_SET_ADDRESS_MODE | Set data order for transfers from host to peripheral. |
| MIPI_DSI_DCS_ID_SET_ARP_OFF | Disable ARP. |
| MIPI_DSI_DCS_ID_SET_ARP_ON | Enable ARP and set T2 timer. |

| | |
|---|---|
| MIPI_DSI_DCS_ID_SET_CABC_MIN_BRIGHTNESS | Set minimum brightness level for CABC mode. |
| MIPI_DSI_DCS_ID_SET_COLUMN_ADDRESS | Set column extent. |
| MIPI_DSI_DCS_ID_SET_DISPLAY_BRIGHTNESS | Set display brightness level. |
| MIPI_DSI_DCS_ID_SET_DISPLAY_OFF | Blank the display device. |
| MIPI_DSI_DCS_ID_SET_DISPLAY_ON | Show image on display device. |
| MIPI_DSI_DCS_ID_SET_DSI_MODE | Set DSI operation mode. |
| MIPI_DSI_DCS_ID_SET_GAMMA_CURVE | Select gamma curve used by display. |
| MIPI_DSI_DCS_ID_SET_PAGE_ADDRESS | Set page extent. |
| MIPI_DSI_DCS_ID_SET_PARTIAL_COLUMNS | Define the number of columns in the partial display area. |
| MIPI_DSI_DCS_ID_SET_PARTIAL_ROWS | Define the number of rows in the partial display area. |
| MIPI_DSI_DCS_ID_SET_PIXEL_FORMAT | Define how many bits per pixel are used. |
| MIPI_DSI_DCS_ID_SET_SCROLL_AREA | Define vertical scrolling and fixed area. |
| MIPI_DSI_DCS_ID_SET_SCROLL_START | Define vertical scrolling starting point. |
| MIPI_DSI_DCS_ID_SET_TEAR_OFF | Sync information not sent from the display module to the host. |
| MIPI_DSI_DCS_ID_SET_TEAR_ON | Sync information is sent from the display module to the host. |
| MIPI_DSI_DCS_ID_SET_TEAR_SCANLINE | Sync information is sent from display to the host when display refresh reaches profivided scan line. |
| MIPI_DSI_DCS_ID_SET_VSYNC_TIMING | Set VSYNC timing to the specified length of PPS data. |
| MIPI_DSI_DCS_ID_SOFT_RESET | Software reset. |
| MIPI_DSI_DCS_ID_WRITE_ACMD | Write access to ACMD registers. |
| MIPI_DSI_DCS_ID_WRITE_CONTROL_DISPLAY | Write control mode of display brightness. |
| MIPI_DSI_DCS_ID_WRITE_DSE_MAILBOX | Write registers of DSE read or write control mailbox. |

| | |
|---|---|
| MIPI_DSI_DCS_ID_WRITE_LUT | Fill peripheral look-up table with provided data. |
| MIPI_DSI_DCS_ID_WRITE_MEMORY_CONTINUE | Continue image information transfer from last address. |
| MIPI_DSI_DCS_ID_WRITE_MEMORY_START | Transfer image information from host to peripheral. |
| MIPI_DSI_DCS_ID_WRITE_POWER_SAVE | Writes power save mode. |

### ◆ mipi_dsi_video_data_t

| enum mipi_dsi_video_data_t | |
|---|---|
| MIPI DSI Video Data type | |
| Enumerator | |
| MIPI_DSI_VIDEO_DATA_16RGB_PIXEL_STREAM | 16-bit RGB Packed Pixel Stream |
| MIPI_DSI_VIDEO_DATA_18RGB_PIXEL_STREAM | 18-bit RGB Packed Pixel Stream |
| MIPI_DSI_VIDEO_DATA_18RGB_LOOSELY_PIXEL_STREAM | 18-bit RGB Loosely Packed Pixel Stream |
| MIPI_DSI_VIDEO_DATA_24RGB_PIXEL_STREAM | 24-bit RGB Packed Pixel Stream |

### ◆ mipi_dsi_ack_err_t

| enum mipi_dsi_ack_err_t | |
|---|---|
| MIPI DSI Acknowledge and Error type | |
| Enumerator | |
| MIPI_DSI_ACK_ERR_NONE | No Errors. |
| MIPI_DSI_ACK_ERR_SOT_ERROR | SoT Error. |
| MIPI_DSI_ACK_ERR_SOT_SYNC_ERROR | SoT Sync Error. |
| MIPI_DSI_ACK_ERR_EOT_SYNC_ERROR | EoT Sync Error. |
| MIPI_DSI_ACK_ERR_ESCAPE_ENTRY_ERROR | Escape Mode Entry Error. |
| MIPI_DSI_ACK_ERR_LOW_POWER_SYNC_ERROR | Low-Power Transmit Sync Error. |
| MIPI_DSI_ACK_ERR_PERIPHERAL_TIMEOUT_ERRO R | Peripheral Timeout Error. |
| MIPI_DSI_ACK_ERR_FALSE_CONTROL_ERROR | False Control Error. |
| MIPI_DSI_ACK_ERR_CONTENTION_DETECTED | Contention Detected Error. |
| MIPI_DSI_ACK_ERR_ECC_SINGLE | ECC Error, single-bit. |
| MIPI_DSI_ACK_ERR_ECC_MULTI | ECC Error, multi-bit. |
| MIPI_DSI_ACK_ERR_CKSM_ERROR | Checksum Error (Long packet only) |
| MIPI_DSI_ACK_ERR_DSI_DATA_ERROR | DSI Data Type Not Recognized. |
| MIPI_DSI_ACK_ERR_DSI_VC_ID_ERROR | DSI VC ID Invalid. |
| MIPI_DSI_ACK_ERR_INVALID_TX_LEN | Invalid Transmission Length. |
| MIPI_DSI_ACK_ERR_DSI_PROTOCOL_VIOLATION | DSI Protocol Violation. |

### ◆ mipi_dsi_vc_t

| enum mipi_dsi_vc_t | |
|---|---|
| Enumerator | |
| MIPI_DSI_VC_NONE | No channels selected. |
| MIPI_DSI_VC_0 | Virtual channel 0. |
| MIPI_DSI_VC_1 | Virtual channel 1. |
| MIPI_DSI_VC_2 | Virtual channel 2. |
| MIPI_DSI_VC_3 | Virtual channel 3. |

### ◆ mipi_dsi_cmd_flag_t

| enum mipi_dsi_cmd_flag_t | |
|---|---|
| MIPI DSI Message Flags | |
| Enumerator | |
| MIPI_DSI_CMD_FLAG_NONE | No flags. |
| MIPI_DSI_CMD_FLAG_BTA | Assert bus turnaround at end of transfer. |
| MIPI_DSI_CMD_FLAG_BTA_READ | Assert bus turnaround followed by read request (No WRITE request before BTA) |
| MIPI_DSI_CMD_FLAG_BTA_NO_WRITE | Immediately assert bus turnaround (No WRITE request before BTA) |
| MIPI_DSI_CMD_FLAG_AUX_OPERATION | Execute auxiliary operation command. |
| MIPI_DSI_CMD_FLAG_ACT_CODE_RESET_TRIGGER | Send action code reset trigger message. |
| MIPI_DSI_CMD_FLAG_ACT_CODE_INITIAL_SKEW_CAL | Send action code initial skew calibration message. |
| MIPI_DSI_CMD_FLAG_ACT_CODE_PERIODIC_SKEW_CAL | Send action code periodic skew message. |
| MIPI_DSI_CMD_FLAG_ACT_CODE_NO_OPERATION | Send action code NOOP message. |
| MIPI_DSI_CMD_FLAG_LOW_POWER | Transmit in low-power mode. |

### ◆ mipi_dsi_event_t

| enum mipi_dsi_event_t | |
|---|---|
| MIPI DSI event codes | |
| Enumerator | |
| MIPI_DSI_EVENT_SEQUENCE_0 | Sequence 0 event (Low-Power) |
| MIPI_DSI_EVENT_SEQUENCE_1 | Sequence 1 event (High-Speed) |
| MIPI_DSI_EVENT_VIDEO | Video event. |
| MIPI_DSI_EVENT_RECEIVE | Receive event. |
| MIPI_DSI_EVENT_FATAL | Fatal event. |
| MIPI_DSI_EVENT_PHY | Physical layer event. |
| MIPI_DSI_EVENT_POST_OPEN | Interface has been opened. Perform post-open application processing. |
| MIPI_DSI_EVENT_PRE_START | Video is about to start. Perform pre-video application processing. |

### ◆ mipi_dsi_sequence_status_t

| enum mipi_dsi_sequence_status_t | |
|---|---|
| MIPI DSI Sequence status | |
| Enumerator | |
| MIPI_DSI_SEQUENCE_STATUS_NONE | Sequence status not set. |
| MIPI_DSI_SEQUENCE_STATUS_RUNNING | Sequence operation in progress. |
| MIPI_DSI_SEQUENCE_STATUS_ACTIONS_FINISHED | All descriptor actions finished. |
| MIPI_DSI_SEQUENCE_STATUS_DESCRIPTORS_FINISHED | All descriptors finished. |
| MIPI_DSI_SEQUENCE_STATUS_DESCRIPTOR_ABORT | Descriptor abort interrupt. |
| MIPI_DSI_SEQUENCE_STATUS_SIZE_ERROR | Packet size error. |
| MIPI_DSI_SEQUENCE_STATUS_TX_INTERNAL_BUS_ERROR | Tx internal bus error. |
| MIPI_DSI_SEQUENCE_STATUS_RX_FATAL_ERROR | Receive fatal error. |
| MIPI_DSI_SEQUENCE_STATUS_RX_FAIL | Receive fail. |
| MIPI_DSI_SEQUENCE_STATUS_RX_PACKET_DATA_FAIL | Receive packet data fail. |
| MIPI_DSI_SEQUENCE_STATUS_RX_CORRECTABLE_ERROR | Receive correctable error. |
| MIPI_DSI_SEQUENCE_STATUS_RX_ACK_AND_ERROR | Receive acknowledge and error report. |

### ◆ mipi_dsi_video_status_t

| enum mipi_dsi_video_status_t | |
|---|---|
| MIPI DSI video status errors | |
| Enumerator | |
| MIPI_DSI_VIDEO_STATUS_NONE | Video status not set. |
| MIPI_DSI_VIDEO_STATUS_START | Video started event. |
| MIPI_DSI_VIDEO_STATUS_STOP | Video stopped event. |
| MIPI_DSI_VIDEO_STATUS_RUNNING | Video running status. |
| MIPI_DSI_VIDEO_STATUS_READY | Video ready event. |
| MIPI_DSI_VIDEO_STATUS_TIMING_ERROR | Video timing error event. |
| MIPI_DSI_VIDEO_STATUS_UNDERFLOW | Video buffer underflow event. |
| MIPI_DSI_VIDEO_STATUS_OVERFLOW | Video buffer overflow event. |

### ◆ mipi_dsi_receive_status_t

| enum mipi_dsi_receive_status_t | |
|---|---|
| MIPI DSI receive status errors | |
| Enumerator | |
| MIPI_DSI_RECEIVE_STATUS_NONE | Receive status not set. |
| MIPI_DSI_RECEIVE_STATUS_BTA_REQUEST_END | Receive BTA request end. |
| MIPI_DSI_RECEIVE_STATUS_LP_RX_HOST_TIMEOUT | Receive low power receive timeout. |
| MIPI_DSI_RECEIVE_STATUS_BTA_ACK_TIMEOUT | Receive BTA ack timeout. |
| MIPI_DSI_RECEIVE_STATUS_RESPONSE_PACKET | Receive response. |
| MIPI_DSI_RECEIVE_STATUS_EOTP | Receive end of transmission packet. |
| MIPI_DSI_RECEIVE_STATUS_TEARING_TRIGGER | Receive tearing trigger. |
| MIPI_DSI_RECEIVE_STATUS_ACK_TRIGGER | Receive ack trigger. |
| MIPI_DSI_RECEIVE_STATUS_TEARING_DETECT | Receive tearing detect. |

| | |
|---|---|
| MIPI_DSI_RECEIVE_STATUS_MALFORM_ERROR | Receive malform error. |
| MIPI_DSI_RECEIVE_STATUS_ECC_MULTI | Receive ecc multi-bit error. |
| MIPI_DSI_RECEIVE_STATUS_UNEXPECTED_PACKET | Receive unexpected packet. |
| MIPI_DSI_RECEIVE_STATUS_WORD_COUNT | Receive word count. |
| MIPI_DSI_RECEIVE_STATUS_CRC | Receive crc error. |
| MIPI_DSI_RECEIVE_STATUS_INTERNAL_BUS | Receive internal bus error. |
| MIPI_DSI_RECEIVE_STATUS_BUFFER_OVERFLOW | Receive buffer overflow. |
| MIPI_DSI_RECEIVE_STATUS_TIMEOUT | Receive timeout. |
| MIPI_DSI_RECEIVE_STATUS_NO_RESPONSE | Receive no response. |
| MIPI_DSI_RECEIVE_STATUS_PACKET_SIZE | Receive packet size error. |
| MIPI_DSI_RECEIVE_STATUS_ECC_SINGLE | Receive ecc single bit error. |
| MIPI_DSI_RECEIVE_STATUS_ACK_AND_ERROR | Receive ack and error. |

### ◆ mipi_dsi_fatal_status_t

| enum mipi_dsi_fatal_status_t | |
|---|---|
| MIPI DSI fatal status errors | |
| Enumerator | |
| MIPI_DSI_FATAL_STATUS_NONE | Fatal status not set. |
| MIPI_DSI_FATAL_STATUS_HS_TX_TIMEOUT | Fatal high speed transmit timeout. |
| MIPI_DSI_FATAL_STATUS_LP_RX_TIMEOUT | Fatal low power receive timeout. |
| MIPI_DSI_FATAL_STATUS_BTA_ACK_TIMEOUT | Fatal BTA ack timeout. |
| MIPI_DSI_FATAL_STATUS_ESCAPE_ENTRY_ERROR | Fatal escape mode entry error. |
| MIPI_DSI_FATAL_STATUS_LPDT_SYNC_ERROR | Fatal low power data transmission synchronization error. |
| MIPI_DSI_FATAL_STATUS_CTRL_ERROR | Fatal control error. |
| MIPI_DSI_FATAL_STATUS_LP0_CONTENTION_DETECT | Fatal lane 0 low power contention detect. |
| MIPI_DSI_FATAL_STATUS_LP1_CONTENTION_DETECT | Fatal lane 1 low power contention detect. |
| MIPI_DSI_FATAL_STATUS_LP0_CONTENTION | Fatal lane 0 low power contention status. |
| MIPI_DSI_FATAL_STATUS_LP1_CONTENTION | Fatal lane 1 low power contention status. |

### ◆ mipi_dsi_phy_status_t

| enum mipi_dsi_phy_status_t | |
|---|---|
| MIPI DSI physical lane status | |
| Enumerator | |
| MIPI_DSI_PHY_STATUS_NONE | Physical lane status not set. |
| MIPI_DSI_PHY_STATUS_ULP_NOT_ACTIVE | Physical lane ULP not active. |
| MIPI_DSI_PHY_STATUS_CLOCK_LANE_STOP | Clock lane in stopped state. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_LP_RX | Data lane low power receive mode. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_ULP_RX | Data lane ultra low power receive mode. |

| | |
|---|---|
| MIPI_DSI_PHY_STATUS_DATA_LANE0_NOT_ULP | Data lane 0 not in ULP mode. |
| MIPI_DSI_PHY_STATUS_DATA_LANE1_NOT_ULP | Data lane 1 not in ULP mode. |
| MIPI_DSI_PHY_STATUS_DATA_LANE2_NOT_ULP | Data lane 2 not in ULP mode. |
| MIPI_DSI_PHY_STATUS_DATA_LANE3_NOT_ULP | Data lane 3 not in ULP mode. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_STOP | Data lane 0 stop state. |
| MIPI_DSI_PHY_STATUS_DATA_LANE1_STOP | Data lane 1 stop state. |
| MIPI_DSI_PHY_STATUS_DATA_LANE2_STOP | Data lane 2 stop state. |
| MIPI_DSI_PHY_STATUS_DATA_LANE3_STOP | Data lane 3 stop state. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_RX_TO_TX | Data lane Rx to Tx transition event. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_TX_TO_RX | Data lane Tx to Rx transition event. |
| MIPI_DSI_PHY_STATUS_DATA_LANE0_RX_STATE | Data lane Rx active state. |
| MIPI_DSI_PHY_STATUS_CLOCK_ULPS_ENTER | Clock lane ULPS enter event. |
| MIPI_DSI_PHY_STATUS_CLOCK_ULPS_EXIT | Clock lane ULPS exit event. |
| MIPI_DSI_PHY_STATUS_CLOCK_LP_TO_HS | Clock lane LP to HS transition event. |
| MIPI_DSI_PHY_STATUS_CLOCK_HS_TO_LP | Clock lane HS to LP transition event. |
| MIPI_DSI_PHY_STATUS_DATA_LANE_ULPS_ENTER | Data lane ULPS enter event. |
| MIPI_DSI_PHY_STATUS_DATA_LANE_ULPS_EXIT | Data lane ULPS exit event. |

### ◆ mipi_dsi_link_status_t

| enum mipi_dsi_link_status_t | |
|---|---|
| MIPI DSI link status bits | |
| Enumerator | |
| MIPI_DSI_LINK_STATUS_IDLE | Link idle or uninitialized. |
| MIPI_DSI_LINK_STATUS_CH0_RUNNING | Channel 0 running. |
| MIPI_DSI_LINK_STATUS_CH1_RUNNING | Channel 1 running. |
| MIPI_DSI_LINK_STATUS_VIDEO_RUNNING | Video output running. |
| MIPI_DSI_LINK_STATUS_HP_MODE_BUSY | HP operation busy. |
| MIPI_DSI_LINK_STATUS_LP_MODE_BUSY | LP operation busy. |

### ◆ mipi_dsi_lane_t

| enum mipi_dsi_lane_t | |
|---|---|
| MIPI DSI Lane Type | |
| Enumerator | |
| MIPI_DSI_LANE_CLOCK | Clock Lanes. |
| MIPI_DSI_LANE_DATA_ALL | All Data Lanes. |

## 5.3.7.6 SLCDC Interface

Interfaces » Graphics

### Detailed Description

Interface for Segment LCD controllers.

### Data Structures

| | |
|---|---|
| struct | slcdc_cfg_t |
| struct | slcdc_api_t |
| | |

| | |
|---|---|
| struct | slcdc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | slcdc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | slcdc_bias_method_t |
| enum | slcdc_time_slice_t |
| enum | slcdc_waveform_t |
| enum | slcdc_drive_volt_gen_t |
| enum | slcdc_ref_volt_sel_t |
| enum | slcdc_display_area_control_blink_t |
| enum | slcdc_display_area_t |
| enum | slcdc_contrast_t |
| enum | slcdc_display_on_off_t |
| enum | slcdc_display_enable_disable_t |
| enum | slcdc_display_clock_t |
| enum | slcdc_clk_div_t |

## Data Structure Documentation

### ◆ slcdc_cfg_t

| struct slcdc_cfg_t | | |
|---|---|---|
| SLCDC configuration block | | |
| Data Fields | | |
| slcdc_display_clock_t | slcdc_clock | LCD clock source (LCDSCKSEL) |
| slcdc_clk_div_t | slcdc_clock_setting | LCD clock setting (LCDC0) |
| slcdc_bias_method_t | bias_method | LCD display bias method select (LBAS bit) |
| slcdc_time_slice_t | time_slice | Time slice of LCD display select (LDTY bit) |
| slcdc_waveform_t | waveform | LCD display waveform select (LWAVE bit) |

| | | |
|---|---|---|
| slcdc_drive_volt_gen_t | drive_volt_gen | LCD Drive Voltage Generator Select (MDSET bit) |
| slcdc_contrast_t | contrast | LCD Boost Level (contrast setting) |
| slcdc_ref_volt_sel_t | ref_volt_sel | LCD reference voltage selection (MDSET2 bit) |

### ◆ slcdc_api_t

| struct slcdc_api_t | |
|---|---|
| SLCDC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count) |
| | |
| fsp_err_t(* | modify )(slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data) |
| | |
| fsp_err_t(* | start )(slcdc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(slcdc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | setContrast )(slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast) |
| | |
| fsp_err_t(* | setDisplayArea )(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area) |
| | |
| fsp_err_t(* | close )(slcdc_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)

Open SLCDC.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to display interface control block. Must be declared by user. |
| [in] | p_cfg | Pointer to display configuration structure. All elements of this structure must be set by the user. |

### ◆ write

fsp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count)

Write data to the SLCDC segment data array. Specifies the initial display data. Except when using 8-time slice mode, store values in the lower 4 bits when writing to the A-pattern area and in the upper 4 bits when writing to the B-pattern area.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | start_segment | Specify the start segment number to be written. |
| [in] | p_data | Pointer to the display data to be written to the specified segments. |
| [in] | segment_count | Number of segments to be written. |

### ◆ modify

fsp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data)

Rewrite data in the SLCDC segment data array. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | segment | The segment to be written. |
| [in] | data_mask | Mask the data being displayed. Set 0 to the bit to be rewritten and set 1 to the other bits. Multiple bits can be rewritten. |
| [in] | data | Specify display data to rewrite to the specified segment. |

### ◆ start

fsp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)

Enable display signal output. Displays the segment data on the LCD.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ stop

fsp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)

Disable display signal output. Stops displaying data on the LCD.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ setContrast

fsp_err_t(* slcdc_api_t::setContrast) (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast)

Set the display contrast. This function can be used only when the internal voltage boosting method is used for drive voltage generation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ setDisplayArea

fsp_err_t(* slcdc_api_t::setDisplayArea) (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)

Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to 'blink' the display between A-pattern and B-pattern area data.

When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic IRQ 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual for the detailed procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |
| [in] | display_area | Display area to be used, A-pattern or B-pattern area. |

### ◆ close

fsp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)

Close SLCDC.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to display interface control block. |

### ◆ slcdc_instance_t

struct slcdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| slcdc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| slcdc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |

| | | |
|---|---|---|
| slcdc_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ slcdc_ctrl_t

| typedef void slcdc_ctrl_t |
|---|
| SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls.SLCDC control block |

### Enumeration Type Documentation

#### ◆ slcdc_bias_method_t

| enum slcdc_bias_method_t | |
|---|---|
| LCD display bias method. | |
| Enumerator | |
| SLCDC_BIAS_2 | 1/2 bias method |
| SLCDC_BIAS_3 | 1/3 bias method |
| SLCDC_BIAS_4 | 1/4 bias method |

#### ◆ slcdc_time_slice_t

| enum slcdc_time_slice_t | |
|---|---|
| Time slice of LCD display. | |
| Enumerator | |
| SLCDC_STATIC | Static. |
| SLCDC_SLICE_2 | 2-time slice |
| SLCDC_SLICE_3 | 3-time slice |
| SLCDC_SLICE_4 | 4-time slice |
| SLCDC_SLICE_6 | 6-time slice |
| SLCDC_SLICE_8 | 8-time slice |

### ◆ slcdc_waveform_t

| enum slcdc_waveform_t | |
|---|---|
| LCD display waveform select. | |
| Enumerator | |
| SLCDC_WAVE_A | Waveform A. |
| SLCDC_WAVE_B | Waveform B. |

### ◆ slcdc_drive_volt_gen_t

| enum slcdc_drive_volt_gen_t | |
|---|---|
| LCD Drive Voltage Generator Select. | |
| Enumerator | |
| SLCDC_VOLT_EXTERNAL | External resistance division method. |
| SLCDC_VOLT_INTERNAL | Internal voltage boosting method. |
| SLCDC_VOLT_CAPACITOR | Capacitor split method. |

### ◆ slcdc_ref_volt_sel_t

| enum slcdc_ref_volt_sel_t | |
|---|---|
| LCD Reference Voltage Selection. | |
| Enumerator | |
| SLCDC_REF_INTERNAL_VL1_CAPACITOR_VCC_EXTERNAL | Select VL1 reference for internal voltage or VCC reference for capacitor split or external division.<br><br>Select VL2 reference for internal voltage or VL4 reference for capacitor split |

### ◆ slcdc_display_area_control_blink_t

| enum slcdc_display_area_control_blink_t | |
|---|---|
| Display Data Area Control | |
| Enumerator | |
| SLCDC_NOT_BLINKING | Display either A-pattern or B-pattern data. |
| SLCDC_BLINKING | Alternately display A-pattern and B-pattern data. |

### ◆ slcdc_display_area_t

| enum slcdc_display_area_t | |
|---|---|
| Display Area data | |
| Enumerator | |
| SLCDC_DISP_A | Display A-pattern data. |
| SLCDC_DISP_B | Display B-pattern data. |
| SLCDC_DISP_BLINK | Blink between A- and B-pattern. |

### ◆ slcdc_contrast_t

| enum slcdc_contrast_t | |
|---|---|
| LCD Boost Level (contrast) settings | |
| Enumerator | |
| SLCDC_CONTRAST_0 | Contrast level 0. |
| SLCDC_CONTRAST_1 | Contrast level 1. |
| SLCDC_CONTRAST_2 | Contrast level 2. |
| SLCDC_CONTRAST_3 | Contrast level 3. |
| SLCDC_CONTRAST_4 | Contrast level 4. |
| SLCDC_CONTRAST_5 | Contrast level 5. |
| SLCDC_CONTRAST_6 | Contrast level 6. |
| | |

| | |
|---|---|
| SLCDC_CONTRAST_7 | Contrast level 7. |
| SLCDC_CONTRAST_8 | Contrast level 8. |
| SLCDC_CONTRAST_9 | Contrast level 9. |
| SLCDC_CONTRAST_10 | Contrast level 10. |
| SLCDC_CONTRAST_11 | Contrast level 11. |
| SLCDC_CONTRAST_12 | Contrast level 12. |
| SLCDC_CONTRAST_13 | Contrast level 13. |
| SLCDC_CONTRAST_14 | Contrast level 14. |
| SLCDC_CONTRAST_15 | Contrast level 15. |
| SLCDC_CONTRAST_16 | Contrast level 16. |
| SLCDC_CONTRAST_17 | Contrast level 17. |
| SLCDC_CONTRAST_18 | Contrast level 18. |
| SLCDC_CONTRAST_19 | Contrast level 19. |
| SLCDC_CONTRAST_20 | Contrast level 20. |
| SLCDC_CONTRAST_21 | Contrast level 21. |
| SLCDC_CONTRAST_22 | Contrast level 22. |

### ◆ slcdc_display_on_off_t

| enum slcdc_display_on_off_t | |
|---|---|
| LCD Display Enable/Disable | |
| Enumerator | |
| SLCDC_DISP_OFF | Display off. |
| SLCDC_DISP_ON | Display on. |

### ◆ slcdc_display_enable_disable_t

| enum slcdc_display_enable_disable_t | |
|---|---|
| LCD Display output enable | |
| Enumerator | |
| SLCDC_DISP_DISABLE | Output ground level to segment/common pins. |
| SLCDC_DISP_ENABLE | Output enable. |

### ◆ slcdc_display_clock_t

| enum slcdc_display_clock_t | |
|---|---|
| LCD Display clock selection | |
| Enumerator | |
| SLCDC_CLOCK_LOCO | Display clock source LOCO. |
| SLCDC_CLOCK_SOSC | Display clock source SOSC. |
| SLCDC_CLOCK_MOSC | Display clock source MOSC. |
| SLCDC_CLOCK_MOCO | Display clock source MOCO. |
| SLCDC_CLOCK_HOCO | Display clock source HOCO. |

### ◆ slcdc_clk_div_t

| enum slcdc_clk_div_t | |
|---|---|
| LCD clock settings | |
| Enumerator | |
| SLCDC_CLK_DIVISOR_LOCO_4 | LOCO Clock/4. |
| SLCDC_CLK_DIVISOR_LOCO_8 | LOCO Clock/8. |
| SLCDC_CLK_DIVISOR_LOCO_16 | LOCO Clock/16. |
| SLCDC_CLK_DIVISOR_LOCO_32 | LOCO Clock/32. |
| SLCDC_CLK_DIVISOR_LOCO_64 | LOCO Clock/64. |
| SLCDC_CLK_DIVISOR_LOCO_128 | LOCO Clock/128. |

| | |
|---|---|
| SLCDC_CLK_DIVISOR_LOCO_256 | LOCO Clock/256. |
| SLCDC_CLK_DIVISOR_LOCO_512 | LOCO Clock/512. |
| SLCDC_CLK_DIVISOR_LOCO_1024 | LOCO Clock/1024. |
| SLCDC_CLK_DIVISOR_HOCO_256 | HOCO Clock/256. |
| SLCDC_CLK_DIVISOR_HOCO_512 | HOCO Clock/512. |
| SLCDC_CLK_DIVISOR_HOCO_1024 | HOCO Clock/1024. |
| SLCDC_CLK_DIVISOR_HOCO_2048 | HOCO Clock/2048. |
| SLCDC_CLK_DIVISOR_HOCO_4096 | HOCO Clock/4096. |
| SLCDC_CLK_DIVISOR_HOCO_8192 | HOCO Clock/8192. |
| SLCDC_CLK_DIVISOR_HOCO_16384 | HOCO Clock/16384. |
| SLCDC_CLK_DIVISOR_HOCO_32768 | HOCO Clock/32768. |
| SLCDC_CLK_DIVISOR_HOCO_65536 | HOCO Clock/65536. |
| SLCDC_CLK_DIVISOR_HOCO_131072 | HOCO Clock/131072. |
| SLCDC_CLK_DIVISOR_HOCO_262144 | HOCO Clock/262144. |
| SLCDC_CLK_DIVISOR_HOCO_524288 | HOCO Clock/524288. |
| SLCDC_CLK_DIVISOR_HOCO_1048576 | HOCO Clock/1048576. |

## 5.3.8 Input

Interfaces

### Detailed Description

Input Interfaces.

### Modules

External IRQ Interface

Interface for detecting external interrupts.

| | |
|---|---|
| | Key Matrix Interface<br>Interface for key matrix functions. |

### 5.3.8.1 External IRQ Interface

Interfaces » Input

#### Detailed Description

Interface for detecting external interrupts.

## Summary

The External IRQ Interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

#### Data Structures

| | |
|---|---|
| struct | external_irq_callback_args_t |
| struct | external_irq_cfg_t |
| struct | external_irq_api_t |
| struct | external_irq_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | external_irq_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | external_irq_trigger_t |
| enum | external_irq_clock_source_div_t |

#### Data Structure Documentation

◆ **external_irq_callback_args_t**

| struct external_irq_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in external_irq_api_t::open |

| | | |
|---|---|---|
| | | function in external_irq_cfg_t. |
| uint32_t | channel | The physical hardware channel that caused the interrupt. |

### ◆ external_irq_cfg_t

| struct external_irq_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint8_t | channel |
| | Hardware channel used. |
| | |
| uint8_t | ipl |
| | Interrupt priority. |
| | |
| IRQn_Type | irq |
| | Interrupt number assigned to this instance. |
| | |
| external_irq_trigger_t | trigger |
| | Trigger setting. |
| | |
| external_irq_clock_source_div_t | clock_source_div |
| | Digital filter clock divisor setting. |
| | |
| bool | filter_enable |
| | Digital filter enable/disable selection. |
| | |
| void(* | p_callback )(external_irq_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| void const * | p_extend |

| | |
|---|---|
| | External IRQ hardware dependent configuration. |

## Field Documentation

### ◆ p_callback

void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)

Callback provided external input trigger occurs.

### ◆ p_context

void* external_irq_cfg_t::p_context

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

### ◆ external_irq_api_t

struct external_irq_api_t

External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg) |
| fsp_err_t(* | enable )(external_irq_ctrl_t *const p_ctrl) |
| fsp_err_t(* | disable )(external_irq_ctrl_t *const p_ctrl) |
| fsp_err_t(* | callbackSet )(external_irq_ctrl_t *const p_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory) |
| fsp_err_t(* | close )(external_irq_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of the structure must be set by user. |

### ◆ enable

fsp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)

Enable callback when an external trigger condition occurs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ disable

fsp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)

Disable callback when external trigger condition occurs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ callbackSet

fsp_err_t(* external_irq_api_t::callbackSet) (external_irq_ctrl_t *const p_ctrl, void(*p_callback)(external_irq_callback_args_t *), void *const p_context, external_irq_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the External IRQ control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in Open call for this external interrupt. |

### ◆ external_irq_instance_t

struct external_irq_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| external_irq_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| external_irq_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| external_irq_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ external_irq_ctrl_t

| typedef void external_irq_ctrl_t |
| --- |
| External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls. |

## Enumeration Type Documentation

### ◆ external_irq_trigger_t

| enum external_irq_trigger_t | |
| --- | --- |
| Condition that will trigger an interrupt when detected. | |
| Enumerator | |
| EXTERNAL_IRQ_TRIG_FALLING | Falling edge trigger. |
| EXTERNAL_IRQ_TRIG_RISING | Rising edge trigger. |
| EXTERNAL_IRQ_TRIG_BOTH_EDGE | Both edges trigger. |
| EXTERNAL_IRQ_TRIG_LEVEL_LOW | Low level trigger. |
| EXTERNAL_IRQ_TRIG_LEVEL_HIGH | High level trigger. |

### ◆ external_irq_clock_source_div_t

| enum external_irq_clock_source_div_t | |
| --- | --- |
| External IRQ input pin digital filtering sample clock divisor settings. The digital filter rejects trigger conditions that are shorter than 3 periods of the filter clock. | |
| Enumerator | |
| EXTERNAL_IRQ_CLOCK_SOURCE_DIV_1 | Filter using clock source divided by 1. |
| EXTERNAL_IRQ_CLOCK_SOURCE_DIV_8 | Filter using clock source divided by 8. |
| EXTERNAL_IRQ_CLOCK_SOURCE_DIV_32 | Filter using clock source divided by 32. |
| EXTERNAL_IRQ_CLOCK_SOURCE_DIV_64 | Filter using clock source divided by 64. |

### 5.3.8.2 Key Matrix Interface

Interfaces » Input

### Detailed Description

Interface for key matrix functions.

# Summary

The KEYMATRIX interface provides standard Key Matrix functionality including event generation on a rising or falling edge for one or more channels at the same time. The generated event indicates all channels that are active in that instant via a bit mask. This allows the interface to be used with a matrix configuration or a one-to-one hardware implementation that is triggered on either a rising or a falling edge.

### Data Structures

| | |
|---|---|
| struct | keymatrix_callback_args_t |
| struct | keymatrix_cfg_t |
| struct | keymatrix_api_t |
| struct | keymatrix_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | keymatrix_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | keymatrix_trigger_t |

### Data Structure Documentation

#### ◆ keymatrix_callback_args_t

| struct keymatrix_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t. |
| uint32_t | channel_mask | Bit vector representing the physical hardware channel(s) that caused the interrupt. |

#### ◆ keymatrix_cfg_t

| struct keymatrix_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| | |

| | |
|---|---|
| uint32_t | channel_mask |
| | Key Input channel(s). Bit mask of channels to open. |
| | |
| keymatrix_trigger_t | trigger |
| | Key Input trigger setting. |
| | |
| uint8_t | ipl |
| | Interrupt priority level. |
| | |
| IRQn_Type | irq |
| | NVIC IRQ number. |
| | |
| void(* | p_callback )(keymatrix_callback_args_t *p_args) |
| | Callback for key interrupt ISR. |
| | |
| void * | p_context |
| | Holder for user data. Passed to callback in keymatrix_user_cb_data_t. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ keymatrix_api_t

| struct keymatrix_api_t | |
|---|---|
| Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | enable )(keymatrix_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | disable )(keymatrix_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(keymatrix_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to control block. Must be declared by user. Value set in this function. |
| [in] | p_cfg | Pointer to configuration structure. All elements of the structure must be set by user. |

### ◆ enable

fsp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)

Enable Key interrupt

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block pointer set in Open call for this Key interrupt. |

### ◆ disable

fsp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)

Disable Key interrupt.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block pointer set in Open call for this Key interrupt. |

### ◆ close

| fsp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl) |
|---|
| Allow driver to be reconfigured. May reduce power consumption.<br><br>**Parameters**<br>[in] p_ctrl — Control block pointer set in Open call for this Key interrupt. |

### ◆ keymatrix_instance_t

| struct keymatrix_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| keymatrix_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| keymatrix_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| keymatrix_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ keymatrix_ctrl_t

| typedef void keymatrix_ctrl_t |
|---|
| Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls. |

## Enumeration Type Documentation

### ◆ keymatrix_trigger_t

| enum keymatrix_trigger_t | |
|---|---|
| Trigger type: rising edge, falling edge | |
| Enumerator | |
| KEYMATRIX_TRIG_FALLING | Falling edge trigger. |
| KEYMATRIX_TRIG_RISING | Rising edge trigger. |

## 5.3.9 Monitoring

Interfaces

### Detailed Description

Monitoring Interfaces.

### Modules

| | |
|---|---|
| CAC Interface | Interface for clock frequency accuracy measurements. |
| CRC Interface | Interface for cyclic redundancy checking. |
| DOC Interface | Interface for the Data Operation Circuit. |
| Low Voltage Detection Interface | Interface for Low Voltage Detection. |
| WDT Interface | Interface for watch dog timer functions. |

## 5.3.9.1 CAC Interface

Interfaces » Monitoring

### Detailed Description

Interface for clock frequency accuracy measurements.

# Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of pulses of the clock to be measured.

## Data Structures

| | |
|---|---|
| struct | cac_ref_clock_config_t |
| struct | cac_meas_clock_config_t |
| struct | cac_callback_args_t |
| struct | cac_cfg_t |
| struct | cac_api_t |
| struct | cac_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | cac_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | cac_event_t |
| enum | cac_clock_type_t |
| enum | cac_clock_source_t |
| enum | cac_ref_divider_t |
| enum | cac_ref_digfilter_t |
| enum | cac_ref_edge_t |
| enum | cac_meas_divider_t |

## Data Structure Documentation

### ◆ cac_ref_clock_config_t

| struct cac_ref_clock_config_t | | |
|---|---|---|
| Structure defining the settings that apply to reference clock configuration. | | |
| Data Fields | | |
| cac_ref_divider_t | divider | Divider specification for the Reference clock. |
| cac_clock_source_t | clock | Clock source for the Reference clock. |
| cac_ref_digfilter_t | digfilter | Digital filter selection for the CACREF ext clock. |
| cac_ref_edge_t | edge | Edge detection for the |

| | | |
|---|---|---|
| | | Reference clock. |

### ◆ cac_meas_clock_config_t

| struct cac_meas_clock_config_t | | |
|---|---|---|
| Structure defining the settings that apply to measurement clock configuration. | | |
| Data Fields | | |
| cac_meas_divider_t | divider | Divider specification for the Measurement clock. |
| cac_clock_source_t | clock | Clock source for the Measurement clock. |

### ◆ cac_callback_args_t

| struct cac_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| cac_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Value provided in configuration structure. |

### ◆ cac_cfg_t

| struct cac_cfg_t | |
|---|---|
| CAC Configuration | |
| **Data Fields** | |
| cac_ref_clock_config_t | cac_ref_clock |
| | Reference clock specific settings. |
| | |
| cac_meas_clock_config_t | cac_meas_clock |
| | Measurement clock specific settings. |
| | |
| uint16_t | cac_upper_limit |
| | The upper limit counter threshold. |
| | |
| uint16_t | cac_lower_limit |
| | The lower limit counter threshold. |

| | |
|---|---|
| | |
| IRQn_Type | mendi_irq |
| | Measurement End IRQ number. |
| | |
| IRQn_Type | ovfi_irq |
| | Measurement Overflow IRQ number. |
| | |
| IRQn_Type | ferri_irq |
| | Frequency Error IRQ number. |
| | |
| uint8_t | mendi_ipl |
| | Measurement end interrupt priority. |
| | |
| uint8_t | ovfi_ipl |
| | Overflow interrupt priority. |
| | |
| uint8_t | ferri_ipl |
| | Frequency error interrupt priority. |
| | |
| void(* | p_callback )(cac_callback_args_t *p_args) |
| | Callback provided when a CAC interrupt ISR occurs. |
| | |
| void * | p_context |
| | Passed to user callback in cac_callback_args_t. |
| | |
| void const * | p_extend |
| | CAC hardware dependent configuration */. |
| | |

### ◆ cac_api_t

| struct cac_api_t | |
|---|---|
| CAC functions implemented at the HAL layer API | |
| **Data Fields** | |
| fsp_err_t(* | open )(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | startMeasurement )(cac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stopMeasurement )(cac_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(cac_ctrl_t *const p_ctrl, uint32_t *const p_counter) |
| | |
| fsp_err_t(* | callbackSet )(cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(cac_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to CAC device control. Must be declared by user. |
| [in] | cac_cfg_t | Pointer to CAC configuration structure. |

### ◆ startMeasurement

fsp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ stopMeasurement

fsp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)

End a measurement for the CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ read

fsp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint32_t *const p_counter)

Read function for CAC peripheral.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the CAC device context. |
| [in] | p_counter | Pointer to variable in which to store the current CACNTBR register contents. |

### ◆ callbackSet

fsp_err_t(* cac_api_t::callbackSet) (cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void *const p_context, cac_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in cac_api_t::open call |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)

Close function for CAC device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CAC device control. |

### ◆ cac_instance_t

struct cac_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| cac_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| cac_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| cac_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ cac_ctrl_t

| typedef void cac_ctrl_t |
| --- |
| CAC control block. Allocate an instance specific control block to pass into the CAC API calls. |

## Enumeration Type Documentation

### ◆ cac_event_t

| enum cac_event_t | |
| --- | --- |
| Event types returned by the ISR callback when used in CAC interrupt mode | |
| Enumerator | |
| CAC_EVENT_FREQUENCY_ERROR | Frequency error. |
| CAC_EVENT_MEASUREMENT_COMPLETE | Measurement complete. |
| CAC_EVENT_COUNTER_OVERFLOW | Counter overflow. |

### ◆ cac_clock_type_t

| enum cac_clock_type_t | |
| --- | --- |
| Enumeration of the two possible clocks. | |
| Enumerator | |
| CAC_CLOCK_MEASURED | Measurement clock. |
| CAC_CLOCK_REFERENCE | Reference clock. |

### ◆ cac_clock_source_t

| enum cac_clock_source_t | |
|---|---|
| Enumeration of the possible clock sources for both the reference and measurement clocks. | |
| Enumerator | |
| CAC_CLOCK_SOURCE_MAIN_OSC | Main clock oscillator. |
| CAC_CLOCK_SOURCE_SUBCLOCK | Sub-clock. |
| CAC_CLOCK_SOURCE_HOCO | HOCO (High speed on chip oscillator) |
| CAC_CLOCK_SOURCE_MOCO | MOCO (Middle speed on chip oscillator) |
| CAC_CLOCK_SOURCE_LOCO | LOCO (Low speed on chip oscillator) |
| CAC_CLOCK_SOURCE_PCLKB | PCLKB (Peripheral Clock B) |
| CAC_CLOCK_SOURCE_IWDT | IWDT-dedicated on-chip oscillator. |
| CAC_CLOCK_SOURCE_EXTERNAL | Externally supplied measurement clock on CACREF pin. |

### ◆ cac_ref_divider_t

| enum cac_ref_divider_t | |
|---|---|
| Enumeration of available dividers for the reference clock. | |
| Enumerator | |
| CAC_REF_DIV_32 | Reference clock divided by 32. |
| CAC_REF_DIV_128 | Reference clock divided by 128. |
| CAC_REF_DIV_1024 | Reference clock divided by 1024. |
| CAC_REF_DIV_8192 | Reference clock divided by 8192. |

### ◆ cac_ref_digfilter_t

| enum cac_ref_digfilter_t | |
|---|---|
| Enumeration of available digital filter settings for an external reference clock. | |
| Enumerator | |
| CAC_REF_DIGITAL_FILTER_OFF | No digital filter on the CACREF pin for reference clock. |
| CAC_REF_DIGITAL_FILTER_1 | Sampling clock for digital filter = measuring frequency. |
| CAC_REF_DIGITAL_FILTER_4 | Sampling clock for digital filter = measuring frequency/4. |
| CAC_REF_DIGITAL_FILTER_16 | Sampling clock for digital filter = measuring frequency/16. |

### ◆ cac_ref_edge_t

| enum cac_ref_edge_t | |
|---|---|
| Enumeration of available edge detect settings for the reference clock. | |
| Enumerator | |
| CAC_REF_EDGE_RISE | Rising edge detect for the Reference clock. |
| CAC_REF_EDGE_FALL | Falling edge detect for the Reference clock. |
| CAC_REF_EDGE_BOTH | Both Rising and Falling edges detect for the Reference clock. |

### ◆ cac_meas_divider_t

| enum cac_meas_divider_t | |
|---|---|
| Enumeration of available dividers for the measurement clock | |
| Enumerator | |
| CAC_MEAS_DIV_1 | Measurement clock divided by 1. |
| CAC_MEAS_DIV_4 | Measurement clock divided by 4. |
| CAC_MEAS_DIV_8 | Measurement clock divided by 8. |
| CAC_MEAS_DIV_32 | Measurement clock divided by 32. |

### 5.3.9.2 CRC Interface

Interfaces » Monitoring

#### Detailed Description

Interface for cyclic redundancy checking.

# Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the block using the CPU or by snooping on read or write activity on one of SCI channels.

#### Data Structures

| | |
|---|---|
| struct | crc_input_t |
| struct | crc_cfg_t |
| struct | crc_api_t |
| struct | crc_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | crc_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | crc_polynomial_t |

| | |
|---|---|
| enum | crc_bit_order_t |
| enum | crc_snoop_direction_t |

## Data Structure Documentation

### ◆ crc_input_t

| struct crc_input_t | | |
|---|---|---|
| Structure for CRC inputs | | |
| Data Fields | | |
| uint32_t | num_bytes | Length of input buffer. It must be 4-byte aligned when a 32-bit CRC polynomial function is used. |
| uint32_t | crc_seed | CRC seed value. |
| const void * | p_input_buffer | Pointer to input buffer. |

### ◆ crc_cfg_t

| struct crc_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| uint8_t | channel | Channel number. |
| crc_polynomial_t | polynomial | CRC Generating Polynomial Switching (GPS) |
| crc_bit_order_t | bit_order | CRC Calculation Switching (LMS) |
| uint32_t | snoop_address | Register Snoop Address (CRCSA) |
| void const * | p_extend | CRC Hardware Dependent Configuration. |

### ◆ crc_api_t

| struct crc_api_t | |
|---|---|
| CRC driver structure. General CRC functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(crc_ctrl_t *const p_ctrl) |
| | |
| | |

| | |
|---|---|
| fsp_err_t(* | crcResultGet )(crc_ctrl_t *const p_ctrl, uint32_t *crc_result) |
| | |
| fsp_err_t(* | snoopEnable )(crc_ctrl_t *const p_ctrl, uint32_t crc_seed) |
| | |
| fsp_err_t(* | snoopDisable )(crc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | calculate )(crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

Open the CRC driver module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [in] | p_cfg | Pointer to a configuration structure. |

### ◆ close

fsp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)

Close the CRC module driver

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Configuration was successful. |

### ◆ crcResultGet

fsp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

Return the current calculated value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [out] | crc_result | The calculated value from the last CRC calculation. |

### ◆ snoopEnable

fsp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

Configure and Enable snooping.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [in] | crc_seed | CRC seed. |

### ◆ snoopDisable

fsp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)

Disable snooping.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |

### ◆ calculate

fsp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Perform a CRC calculation on a block of data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to CRC device handle. |
| [in] | p_crc_input | A pointer to structure for CRC inputs |
| [out] | crc_result | The calculated value of the CRC calculation. |

### ◆ crc_instance_t

| struct crc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| crc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| crc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| crc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ crc_ctrl_t

| typedef void crc_ctrl_t |
|---|
| CRC control block. Allocate an instance specific control block to pass into the CRC API calls. |

## Enumeration Type Documentation

### ◆ crc_polynomial_t

| enum crc_polynomial_t | |
|---|---|
| CRC Generating Polynomial Switching (GPS). | |
| Enumerator | |
| CRC_POLYNOMIAL_CRC_8 | 8-bit CRC-8 (X^8 + X^2 + X + 1) |
| CRC_POLYNOMIAL_CRC_16 | 16-bit CRC-16 (X^16 + X^15 + X^2 + 1) |
| CRC_POLYNOMIAL_CRC_CCITT | 16-bit CRC-CCITT (X^16 + X^12 + X^5 + 1) |
| CRC_POLYNOMIAL_CRC_32 | 32-bit CRC-32 (X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1) |
| CRC_POLYNOMIAL_CRC_32C | 32-bit CRC-32C (X^32 + X^28 + X^27 + X^26 + X^25 + X^23 + X^22 + X^20 + X^19 + X^18 + X^14 + X^13 + X^11 + X^10 + X^9 + X^8 + X^6 + 1) |

◆ **crc_bit_order_t**

| enum crc_bit_order_t | |
|---|---|
| CRC Calculation Switching (LMS) | |
| Enumerator | |
| CRC_BIT_ORDER_LMS_LSB | Generates CRC for LSB first communication. |
| CRC_BIT_ORDER_LMS_MSB | Generates CRC for MSB first communication. |

◆ **crc_snoop_direction_t**

| enum crc_snoop_direction_t | |
|---|---|
| Snoop-On-Write/Read Switch (CRCSWR) | |
| Enumerator | |
| CRC_SNOOP_DIRECTION_RECEIVE | Snoop-on-read. |
| CRC_SNOOP_DIRECTION_TRANSMIT | Snoop-on-write. |

### 5.3.9.3 DOC Interface

Interfaces » Monitoring

#### Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

## Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

#### Data Structures

| | |
|---|---|
| struct | doc_callback_args_t |
| struct | doc_cfg_t |
| struct | doc_api_t |

| | |
|---|---|
| struct | doc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | doc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | doc_event_t |
| enum | doc_bit_width_t |

## Data Structure Documentation

### ◆ doc_callback_args_t

| struct doc_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void * | p_context | Set in doc_api_t::open function in doc_cfg_t.<br><br>Placeholder for user data. |

### ◆ doc_cfg_t

| struct doc_cfg_t | |
|---|---|
| User configuration structure, used in the open function. | |
| **Data Fields** | |
| doc_event_t | event |
| | Select enumerated value from doc_event_t. |
| | |
| doc_bit_width_t | bit_width |
| | The bit width of operations. |
| | |
| uint32_t | doc_data |
| | |
| uint32_t | doc_data_extra |
| | |
| uint8_t | ipl |
| | DOC interrupt priority. |

| | |
|---|---|
| | |
| IRQn_Type | irq |
| | Interrupt number assigned to this instance. |
| | |
| void(* | p_callback )(doc_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |

## Field Documentation

### ◆ doc_data

uint32_t doc_cfg_t::doc_data

Initial/Reference data for addition, subtraction, and comparison operations.

- In Addition and Subtraction mode, this value sets the initial value of the operations.
- In Comparison match, mismatch, lower, and upper modes, this value is compared with data that is written.
- In Comparison inside window and outside window modes, this value is used as the lower bound for comparisons.

### ◆ doc_data_extra

uint32_t doc_cfg_t::doc_data_extra

Additional reference data for use in Window Comparison operations.

- In Comparison inside window and outside window modes, this value is used as the upper bound for comparisons.

### ◆ p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Callback provided when a DOC ISR occurs.

### ◆ p_context

void* doc_cfg_t::p_context

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

### ◆ doc_api_t

struct doc_api_t

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(doc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(doc_ctrl_t *const p_ctrl, uint32_t *p_result) |
| | |
| fsp_err_t(* | write )(doc_ctrl_t *const p_ctrl, uint32_t data) |
| | |
| fsp_err_t(* | callbackSet )(doc_ctrl_t *const p_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

fsp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Allow the driver to be reconfigured. Will reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |

### ◆ read

fsp_err_t(* doc_api_t::read) (doc_ctrl_t *const p_ctrl, uint32_t *p_result)

Gets the result of addition/subtraction operations and stores it in the provided pointer p_result.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |
| [in] | p_result | The result of the DOC operation. |

### ◆ write

fsp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, uint32_t data)

Write to the DODIR register.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in doc_api_t::open call. |
| [in] | data | data to be written to DOC DODIR register. |

### ◆ callbackSet

fsp_err_t(* doc_api_t::callbackSet) (doc_ctrl_t *const p_ctrl, void(*p_callback)(doc_callback_args_t *), void *const p_context, doc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the DOC control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ doc_instance_t

struct doc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| doc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| doc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| doc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ doc_ctrl_t

typedef void doc_ctrl_t

DOC control block. Allocate an instance specific control block to pass into the DOC API calls.

## Enumeration Type Documentation

### ◆ doc_event_t

| enum doc_event_t | |
|---|---|
| Event that can trigger a callback function. | |
| Enumerator | |
| DOC_EVENT_COMPARISON_MISMATCH | The data is not equal to the reference data setting. |
| DOC_EVENT_ADDITION | Addition resulted in a value greater than the max for the configured bit width. |
| DOC_EVENT_SUBTRACTION | Subtraction resulted in a value less than 0. |
| DOC_EVENT_COMPARISON_MATCH | The data is equal to the reference data setting. |
| DOC_EVENT_COMPARISON_LOWER | The data is less than the reference data setting. |
| DOC_EVENT_COMPARISON_UPPER | The data is greater than the reference data setting. |
| DOC_EVENT_COMPARISON_INSIDE_WINDOW | The data is between the two reference data settings. |
| DOC_EVENT_COMPARISON_OUTSIDE_WINDOW | The data is outside the two reference data setttings. |

### ◆ doc_bit_width_t

| enum doc_bit_width_t | |
|---|---|
| The bit width used during operations. | |
| Enumerator | |
| DOC_BIT_WIDTH_16 | Operations are 16-bit. |
| DOC_BIT_WIDTH_32 | Operations are 32-bit. |

## 5.3.9.4 Low Voltage Detection Interface

Interfaces » Monitoring

### Detailed Description

Interface for Low Voltage Detection.

# Summary

The LVD driver provides functions for configuring the LVD voltage monitors and detectors.

### Data Structures

| | |
|---|---|
| struct | lvd_status_t |
| struct | lvd_callback_args_t |
| struct | lvd_cfg_t |
| struct | lvd_api_t |
| struct | lvd_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | lvd_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | lvd_threshold_t |
| enum | lvd_response_t |
| enum | lvd_voltage_slope_t |
| enum | lvd_sample_clock_t |
| enum | lvd_negation_delay_t |
| enum | lvd_threshold_crossing_t |
| enum | lvd_current_state_t |

### Data Structure Documentation

#### ◆ lvd_status_t

| struct lvd_status_t | | |
|---|---|---|
| Current state of a voltage monitor. | | |
| Data Fields | | |
| lvd_threshold_crossing_t | crossing_detected | Threshold crossing detection (latched) |

| | | |
|---|---|---|
| lvd_current_state_t | current_state | Instantaneous status of monitored voltage (above or below threshold) |

### ◆ lvd_callback_args_t

| struct lvd_callback_args_t | | |
|---|---|---|
| LVD callback parameter definition | | |
| Data Fields | | |
| uint32_t | monitor_number | Monitor number. |
| lvd_current_state_t | current_state | Current state of the voltage monitor. |
| void * | p_context | Placeholder for user data. |

### ◆ lvd_cfg_t

| struct lvd_cfg_t | |
|---|---|
| LVD configuration structure | |
| **Data Fields** | |
| uint32_t | monitor_number |
| | |
| lvd_threshold_t | voltage_threshold |
| | |
| lvd_response_t | detection_response |
| | |
| lvd_voltage_slope_t | voltage_slope |
| | |
| lvd_negation_delay_t | negation_delay |
| | |
| lvd_sample_clock_t | sample_clock_divisor |
| | |
| IRQn_Type | irq |
| | |
| uint8_t | monitor_ipl |
| | |
| void(* | p_callback )(lvd_callback_args_t *p_args) |
| | |
| void * | p_context |

| | |
|---|---|
| void const * | p_extend |

## Field Documentation

### ◆ monitor_number

uint32_t lvd_cfg_t::monitor_number

Monitor number, 1, 2, ...

### ◆ voltage_threshold

lvd_threshold_t lvd_cfg_t::voltage_threshold

Threshold for out of range voltage detection

### ◆ detection_response

lvd_response_t lvd_cfg_t::detection_response

Response on detecting a threshold crossing

### ◆ voltage_slope

lvd_voltage_slope_t lvd_cfg_t::voltage_slope

Direction of voltage crossing that will trigger a detection (Rising Edge, Falling Edge, Both).

### ◆ negation_delay

lvd_negation_delay_t lvd_cfg_t::negation_delay

Negation of LVD signal follows reset or voltage in range

### ◆ sample_clock_divisor

lvd_sample_clock_t lvd_cfg_t::sample_clock_divisor

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

### ◆ irq

IRQn_Type lvd_cfg_t::irq

Interrupt number.

### ◆ monitor_ipl

uint8_t lvd_cfg_t::monitor_ipl

Interrupt priority level.

### ◆ p_callback

| void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args) |
|---|
| User function to be called from interrupt |

### ◆ p_context

| void* lvd_cfg_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in |

### ◆ p_extend

| void const* lvd_cfg_t::p_extend |
|---|
| Extension parameter for hardware specific settings |

### ◆ lvd_api_t

| struct lvd_api_t | |
|---|---|
| LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | statusGet )(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status) |
| | |
| fsp_err_t(* | statusClear )(lvd_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(lvd_ctrl_t *const p_ctrl, void(*p_callback)(lvd_callback_args_t *), void *const p_context, lvd_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(lvd_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

Initializes a low voltage detection driver according to the passed-in configuration structure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure for the driver instance |
| [in] | p_cfg | Pointer to the configuration structure for the driver instance |

### ◆ statusGet

fsp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)

Get the current state of the monitor, (threshold crossing detected, voltage currently above or below threshold). Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |
| [in,out] | p_lvd_status | Pointer to a lvd_status_t structure |

### ◆ statusClear

fsp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)

Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |

### ◆ callbackSet

fsp_err_t(* lvd_api_t::callbackSet) (lvd_ctrl_t *const p_ctrl, void(*p_callback)(lvd_callback_args_t *), void *const p_context, lvd_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the LVD control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)

Disables the LVD peripheral. Closes the driver instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the control structure for the driver instance |

### ◆ lvd_instance_t

struct lvd_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| lvd_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| lvd_cfg_t const * | p_cfg | Pointer to the configuration structure for this interface instance. |
| lvd_api_t const * | p_api | Pointer to the API structure for this interface instance. |

## Typedef Documentation

### ◆ lvd_ctrl_t

| typedef void lvd_ctrl_t |
| --- |
| LVD control block. Allocate an instance specific control block to pass into the LVD API calls. |

## Enumeration Type Documentation

### ◆ lvd_threshold_t

| enum lvd_threshold_t | |
|---|---|
| Register definitions, common services, and error codes. Voltage detection level The thresholds supported by each MCU are in the MCU User's Manual as well as in the r_lvd module description on the stack tab of the RA project. | |
| Enumerator | |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_08V | 4.08V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_88V | 3.88V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_67V | 3.67V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_47V | 3.47V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_27V | 3.27V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_06V | 3.06V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_91V | 2.91V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_76V | 2.76V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_60V | 2.60V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_45V | 2.45V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_35V | 2.35V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_25V | 2.25V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_15V | 2.15V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_04V | 2.04V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_94V | 1.94V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_84V | 1.84V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_63V | 1.63V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_08V | 4.08V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_88V | 3.88V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_67V | 3.67V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_47V | 3.47V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_27V | 3.27V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_06V | 3.06V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_91V | 2.91V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_76V | 2.76V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_60V | 2.60V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_45V | 2.45V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_35V | 2.35V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_25V | 2.25V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_15V | 2.15V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_04V | 2.04V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_94V | 1.94V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_84V | 1.84V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_63V | 1.63V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_08V | 4.08V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_88V | 3.88V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_67V | 3.67V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_47V | 3.47V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_27V | 3.27V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_06V | 3.06V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_91V | 2.91V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_76V | 2.76V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_60V | 2.60V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_45V | 2.45V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_35V | 2.35V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_25V | 2.25V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_15V | 2.15V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_04V | 2.04V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_94V | 1.94V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_84V | 1.84V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_63V | 1.63V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V | 3.00V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V | 2.90V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V | 2.79V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V | 2.68V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V | 2.58V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V | 2.48V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V | 2.20V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V | 1.96V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V | 1.75V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V | 1.65V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_3_13V | 3.13V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_71V | 2.71V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_50V | 2.50V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_30V | 2.30V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_09V | 2.09V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_1_88V | 1.88V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_1_67V | 1.67V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_3_08V | 3_08V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_78V | 2_78V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_68V | 2_68V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_58V | 2_58V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_38V | 2_38V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_18V | 2_18V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_1_59V | 1_59V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_99V | 2_99V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_79V | 2_79V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_59V | 2_59V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_39V | 2_39V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_19V | 2_19V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_1_99V | 1_99V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_1_79V | 1_79V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_1_59V | 1_59V |

| | |
|---|---|
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V | 3.00V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V | 2.90V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V | 2.79V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V | 2.68V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V | 2.58V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V | 2.48V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V | 2.20V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V | 1.96V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V | 1.75V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V | 1.65V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_3_13V | 3.13V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_71V | 2.71V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_50V | 2.50V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_30V | 2.30V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_09V | 2.09V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_1_88V | 1.88V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_1_67V | 1.67V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_14V | 3.14V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_LEVEL_3_08V | 3.08V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_83V | 2.83V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_80V | 2.80V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_62V | 2.62V |
| LVD_THRESHOLD_MONITOR_LEVEL_2_33V | 2.33V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_90V | 1.90V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_74V | 1.74V |
| LVD_THRESHOLD_MONITOR_LEVEL_1_71V | 1.71V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_29V | 4.29V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_16V | 4.16V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_14V | 4.14V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_03V | 4.03V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_4_02V | 4.02V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_86V | 3.86V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_84V | 3.84V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V | 3.10V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V | 3.00V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V | 2.90V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V | 2.79V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V | 2.68V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V | 2.58V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V | 2.48V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V | 2.20V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V | 1.96V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V | 1.86V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V | 1.75V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V | 1.65V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_99V | 2.99V |

| | |
|---|---|
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_1_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_31V | 4.31V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_29V | 4.29V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_17V | 4.17V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_14V | 4.14V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_03V | 4.03V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_4_02V | 4.02V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_3_84V | 3.84V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_99V | 2.99V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V | 2.92V |
| LVD_THRESHOLD_MONITOR_2_LEVEL_2_85V | 2.85V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_3_1V | 3.1V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_9V | 2.9V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_8V | 2.8V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_7V | 2.7V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_6V | 2.6V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_4V | 2.4V |
| LVD_THRESHOLD_EXLVDVBAT_LEVEL_2_2V | 2.2V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_8V | 2.8V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_6V | 2.6V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_4V | 2.4V |
| LVD_THRESHOLD_LVDVRTC_LEVEL_2_2V | 2.2V |
| LVD_THRESHOLD_NOT_AVAILABLE | Not Used. |

### ◆ lvd_response_t

| enum lvd_response_t | |
|---|---|
| Response types for handling threshold crossing event. | |
| Enumerator | |
| LVD_RESPONSE_NMI | Non-maskable interrupt. |
| LVD_RESPONSE_INTERRUPT | Maskable interrupt. |
| LVD_RESPONSE_RESET | Reset on VCC-fall. |
| LVD_RESPONSE_RESET_ON_RISING | Reset on VCC-rise. |
| LVD_RESPONSE_NONE | No response, status must be requested via statusGet function. |

### ◆ lvd_voltage_slope_t

| enum lvd_voltage_slope_t | |
|---|---|
| The direction from which VCC must cross the threshold to trigger a detection (rising, falling, or both). | |
| Enumerator | |
| LVD_VOLTAGE_SLOPE_RISING | When VCC >= Vdet2 (rise) is detected. |
| LVD_VOLTAGE_SLOPE_FALLING | When VCC < Vdet2 (drop) is detected. |
| LVD_VOLTAGE_SLOPE_BOTH | When drop and rise are detected. |

### ◆ lvd_sample_clock_t

| enum lvd_sample_clock_t | |
|---|---|
| Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering | |
| Enumerator | |
| LVD_SAMPLE_CLOCK_LOCO_DIV_2 | Digital filter sample clock is LOCO divided by 2. |
| LVD_SAMPLE_CLOCK_LOCO_DIV_4 | Digital filter sample clock is LOCO divided by 4. |
| LVD_SAMPLE_CLOCK_LOCO_DIV_8 | Digital filter sample clock is LOCO divided by 8. |
| LVD_SAMPLE_CLOCK_LOCO_DIV_16 | Digital filter sample clock is LOCO divided by 16. |
| LVD_SAMPLE_CLOCK_DISABLED | Digital filter is disabled. |

### ◆ lvd_negation_delay_t

| enum lvd_negation_delay_t | |
|---|---|
| Negation delay of LVD reset signal follows reset or voltage in range | |
| Enumerator | |
| LVD_NEGATION_DELAY_FROM_VOLTAGE | Negation follows a stabilization time (tLVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE |
| LVD_NEGATION_DELAY_FROM_RESET | Negation follows a stabilization time (tLVDn) after assertion of the LVDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE |

### ◆ lvd_threshold_crossing_t

| enum lvd_threshold_crossing_t | |
|---|---|
| Threshold crossing detection (latched) | |
| Enumerator | |
| LVD_THRESHOLD_CROSSING_NOT_DETECTED | Threshold crossing has not been detected. |
| LVD_THRESHOLD_CROSSING_DETECTED | Threshold crossing has been detected. |

### ◆ lvd_current_state_t

| enum lvd_current_state_t | |
|---|---|
| Instantaneous status of VCC (above or below threshold) | |
| Enumerator | |
| LVD_CURRENT_STATE_BELOW_THRESHOLD | VCC < threshold. |
| LVD_CURRENT_STATE_ABOVE_THRESHOLD | VCC >= threshold or monitor is disabled. |

## 5.3.9.5 WDT Interface

Interfaces » Monitoring

### Detailed Description

Interface for watch dog timer functions.

# Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

### Data Structures

| | |
|---|---|
| struct | wdt_callback_args_t |
| struct | wdt_timeout_values_t |
| struct | wdt_cfg_t |
| struct | wdt_api_t |
| struct | wdt_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | wdt_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | wdt_timeout_t |
| enum | wdt_clock_division_t |
| enum | wdt_window_start_t |
| enum | wdt_window_end_t |
| enum | wdt_reset_control_t |
| enum | wdt_stop_control_t |
| enum | wdt_status_t |

### Data Structure Documentation

#### ◆ wdt_callback_args_t

| struct wdt_callback_args_t |
|---|
| Callback function parameter data |
| Data Fields |

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t. |

### ◆ wdt_timeout_values_t

| struct wdt_timeout_values_t | | |
|---|---|---|
| WDT timeout data. Used to return frequency of WDT clock and timeout period | | |
| Data Fields | | |
| uint32_t | clock_frequency_hz | Frequency of watchdog clock after divider. |
| uint32_t | timeout_clocks | Timeout period in units of watchdog clock ticks. |

### ◆ wdt_cfg_t

| struct wdt_cfg_t | |
|---|---|
| WDT configuration parameters. | |
| **Data Fields** | |
| wdt_timeout_t | timeout |
| | Timeout period. |
| | |
| wdt_clock_division_t | clock_division |
| | Clock divider. |
| | |
| wdt_window_start_t | window_start |
| | Refresh permitted window start position. |
| | |
| wdt_window_end_t | window_end |
| | Refresh permitted window end position. |
| | |
| wdt_reset_control_t | reset_control |
| | Select NMI/IRQ or reset generated on underflow. |
| | |
| wdt_stop_control_t | stop_control |
| | Select whether counter operates in sleep mode. |

| | |
|---|---|
| | |
| void(* | p_callback )(wdt_callback_args_t *p_args) |
| | Callback provided when a WDT ISR occurs. |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ p_context

| |
|---|
| void* wdt_cfg_t::p_context |
| Placeholder for user data. Passed to the user callback in wdt_callback_args_t. |

### ◆ wdt_api_t

| | |
|---|---|
| struct wdt_api_t | |
| WDT functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | refresh )(wdt_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status) |
| | |
| fsp_err_t(* | statusClear )(wdt_ctrl_t *const p_ctrl, const wdt_status_t status) |
| | |
| fsp_err_t(* | counterGet )(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count) |
| | |
| fsp_err_t(* | timeoutGet )(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout) |
| | |
| fsp_err_t(* | callbackSet )(wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory) |

## Field Documentation

### ◆ open

fsp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

Initialize the WDT in register start mode. In auto-start mode (Supported devices only) with NMI output it registers the NMI callback.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ refresh

fsp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)

Refresh the watchdog timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ statusGet

fsp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

Read the status of the WDT.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_status | Pointer to variable to return status information through. |

### ◆ statusClear

fsp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

Clear the status flags of the WDT.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | status | Status condition(s) to clear. |

### ◆ counterGet

fsp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)

Read the current WDT counter value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_count | Pointer to variable to return current WDT counter value. |

### ◆ timeoutGet

fsp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

Read the watchdog timeout values.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_timeout | Pointer to structure to return timeout values. |

### ◆ callbackSet

fsp_err_t(* wdt_api_t::callbackSet) (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void *const p_context, wdt_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the WDT control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ wdt_instance_t

struct wdt_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| wdt_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| wdt_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| wdt_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ wdt_ctrl_t

| typedef void wdt_ctrl_t |
|---|
| WDT control block. Allocate an instance specific control block to pass into the WDT API calls. |

## Enumeration Type Documentation

### ◆ wdt_timeout_t

| enum wdt_timeout_t | |
|---|---|
| WDT time-out periods. | |
| Enumerator | |
| WDT_TIMEOUT_128 | 128 clock cycles |
| WDT_TIMEOUT_512 | 512 clock cycles |
| WDT_TIMEOUT_1024 | 1024 clock cycles |
| WDT_TIMEOUT_2048 | 2048 clock cycles |
| WDT_TIMEOUT_4096 | 4096 clock cycles |
| WDT_TIMEOUT_8192 | 8192 clock cycles |
| WDT_TIMEOUT_16384 | 16384 clock cycles |

### ◆ wdt_clock_division_t

| enum wdt_clock_division_t | |
|---|---|
| WDT clock division ratio. | |
| Enumerator | |
| WDT_CLOCK_DIVISION_1 | CLK/1. |
| WDT_CLOCK_DIVISION_4 | CLK/4. |
| WDT_CLOCK_DIVISION_16 | CLK/16. |
| WDT_CLOCK_DIVISION_32 | CLK/32. |
| WDT_CLOCK_DIVISION_64 | CLK/64. |
| WDT_CLOCK_DIVISION_128 | CLK/128. |
| WDT_CLOCK_DIVISION_256 | CLK/256. |
| WDT_CLOCK_DIVISION_512 | CLK/512. |
| WDT_CLOCK_DIVISION_2048 | CLK/2048. |
| WDT_CLOCK_DIVISION_8192 | CLK/8192. |

### ◆ wdt_window_start_t

| enum wdt_window_start_t | |
|---|---|
| WDT refresh permitted period window start position. | |
| Enumerator | |
| WDT_WINDOW_START_25 | Start position = 25%. |
| WDT_WINDOW_START_50 | Start position = 50%. |
| WDT_WINDOW_START_75 | Start position = 75%. |
| WDT_WINDOW_START_100 | Start position = 100%. |

### ◆ wdt_window_end_t

| enum wdt_window_end_t | |
|---|---|
| WDT refresh permitted period window end position. | |
| Enumerator | |
| WDT_WINDOW_END_75 | End position = 75%. |
| WDT_WINDOW_END_50 | End position = 50%. |
| WDT_WINDOW_END_25 | End position = 25%. |
| WDT_WINDOW_END_0 | End position = 0%. |

### ◆ wdt_reset_control_t

| enum wdt_reset_control_t | |
|---|---|
| WDT Counter underflow and refresh error control. | |
| Enumerator | |
| WDT_RESET_CONTROL_NMI | NMI/IRQ request when counter underflows. |
| WDT_RESET_CONTROL_RESET | Reset request when counter underflows. |

### ◆ wdt_stop_control_t

| enum wdt_stop_control_t | |
|---|---|
| WDT Counter operation in sleep mode. | |
| Enumerator | |
| WDT_STOP_CONTROL_DISABLE | Count will not stop when device enters sleep mode. |
| WDT_STOP_CONTROL_ENABLE | Count will automatically stop when device enters sleep mode. |

### ◆ wdt_status_t

| enum wdt_status_t | |
|---|---|
| WDT status | |
| Enumerator | |
| WDT_STATUS_NO_ERROR | No status flags set. |
| WDT_STATUS_UNDERFLOW | Underflow flag set. |
| WDT_STATUS_REFRESH_ERROR | Refresh error flag set. Refresh outside of permitted window. |
| WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR | Underflow and refresh error flags set. |
| WDT_STATUS_OVERFLOW | Overflow flag set. |

## 5.3.10 Motor

Interfaces

### Detailed Description

Motor Interfaces.

### Modules

| | |
|---|---|
| | Motor 120-Degree Control Interface<br>Interface for motor 120 control functions. |
| | Motor 120-Degree Driver Interface<br>Interface for motor driver functions. |
| | Motor Inertia Estimate Interface<br>Interface for Motor inertia estimate functions. |
| | Motor Interface<br>Interface for Motor functions. |

| | |
|---|---|
| | Motor Return Origin Function Interface<br>Interface for Motor return origin functions. |
| | Motor angle Interface<br>Interface for motor angle and speed calculation functions. |
| | Motor current Interface<br>Interface for motor current functions. |
| | Motor driver Interface<br>Interface for motor driver functions. |
| | Motor position Interface<br>Interface for motor position functions. |
| | Motor speed Interface<br>Interface for motor speed functions. |

### 5.3.10.1 Motor 120-Degree Control Interface

Interfaces » Motor

#### Detailed Description

Interface for motor 120 control functions.

## Summary

The motor 120 control interface for speed calculation and setting, fixed cycle processing

#### Data Structures

| | |
|---|---|
| struct | motor_120_control_callback_args_t |
| struct | motor_120_control_motor_parameter_t |
| struct | motor_120_control_cfg_t |
| | |

| | |
|---|---|
| struct | motor_120_control_api_t |
| struct | motor_120_control_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_120_control_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_120_control_event_t |
| enum | motor_120_conduction_type_t |
| enum | motor_120_control_status_t |
| enum | motor_120_control_run_mode_t |
| enum | motor_120_control_rotation_direction_t |
| enum | motor_120_control_wait_stop_flag_t |
| enum | motor_120_control_timeout_error_flag_t |
| enum | motor_120_control_pattern_error_flag_t |
| enum | motor_120_control_speed_ref_t |
| enum | motor_120_control_voltage_ref_t |

## Data Structure Documentation

### ◆ motor_120_control_callback_args_t

| struct motor_120_control_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| motor_120_control_event_t | event | Event trigger. |
| void * | p_context | Placeholder for user data. |

### ◆ motor_120_control_motor_parameter_t

| struct motor_120_control_motor_parameter_t | | |
|---|---|---|
| Motor parameter for motor 120 control | | |
| Data Fields | | |
| uint32_t | u4_motor_pp | Pole pairs. |
| float | f4_motor_r | Resistance (ohm) |

| | | |
|---|---|---|
| float | f4_motor_ld | Inductance for d-axis (H) |
| float | f4_motor_lq | Inductance for q-axis (H) |
| float | f4_motor_m | Magnet flux (Wb) |
| float | f4_motor_j | Rotor inertia (kgm^2) |

### ◆ motor_120_control_cfg_t

| struct motor_120_control_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| motor_120_conduction_type_t | conduction_type |
| | 0:First 60 degree PWM, 1:Complementary first 60 degree PWM |
| | |
| uint32_t | u4_timeout_cnt |
| | Undetected time. |
| | |
| float | f4_max_drive_v |
| | Maximum output voltage (V) |
| | |
| float | f4_min_drive_v |
| | Minimum output voltage (V) |
| | |
| uint32_t | u4_speed_pi_decimation |
| | Speed PI control decimation counter. |
| | |
| uint32_t | u4_free_run_timer_freq |
| | Speed calc free run timer frequency (MHz) |
| | |
| float | f4_speed_lpf_k |
| | Speed LPF parameter. |
| | |

| | |
|---|---|
| | |
| float | f4_limit_speed_change |
| | Speed ref change limit. |
| | |
| float | f4_pi_ctrl_kp |
| | PI control error. |
| | |
| float | f4_pi_ctrl_ki |
| | PI control buffer of integral term. |
| | |
| float | f4_pi_ctrl_ilimit |
| | PI control limit of integral term. |
| | |
| motor_120_control_motor_parameter_t | motor_param |
| | Motor parameter. |
| | |
| void(* | p_callback )(motor_120_control_callback_args_t *p_args) |
| | Callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Extended configurations. |
| | |

### ◆ motor_120_control_api_t

| |
|---|
| struct motor_120_control_api_t |
| Functions implemented at the HAL layer will follow these APIs. |

| Data Fields | |
|---|---|
| fsp_err_t(* | open )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_120_control_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | run )(motor_120_control_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(motor_120_control_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_120_control_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | speedSet )(motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm) |
| | |
| fsp_err_t(* | speedGet )(motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm) |
| | |
| fsp_err_t(* | currentGet )(motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status) |
| | |
| fsp_err_t(* | waitStopFlagGet )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag) |
| | |
| fsp_err_t(* | timeoutErrorFlagGet )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag) |
| | |
| fsp_err_t(* | patternErrorFlagGet )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag) |
| | |
| fsp_err_t(* | voltageRefGet )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_120_control_api_t::open) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)

Initialize the motor 120 control module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_120_control_api_t::close) (motor_120_control_ctrl_t *const p_ctrl)

Close the motor 120 control module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ run

fsp_err_t(* motor_120_control_api_t::run) (motor_120_control_ctrl_t *const p_ctrl)

Run the motor 120 control module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stop

fsp_err_t(* motor_120_control_api_t::stop) (motor_120_control_ctrl_t *const p_ctrl)

Stop the motor 120 control module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_120_control_api_t::reset) (motor_120_control_ctrl_t *const p_ctrl)

Reset variables of the motor 120 control module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ speedSet

fsp_err_t(* motor_120_control_api_t::speedSet) (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm)

Set speed[rpm]

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | speed_rpm | Pointer to get speed data[rpm] |

### ◆ speedGet

fsp_err_t(* motor_120_control_api_t::speedGet) (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm)

Get speed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_speed_rpm | Pointer to get speed data[rpm] |

### ◆ currentGet

fsp_err_t(* motor_120_control_api_t::currentGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)

Get phase current, Vdc and Va_max data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_current_status | Pointer to get data structure. |

### ◆ waitStopFlagGet

fsp_err_t(* motor_120_control_api_t::waitStopFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag)

Get wait stop flag.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_flag | Pointer to wait stop flag |

### ◆ timeoutErrorFlagGet

fsp_err_t(* motor_120_control_api_t::timeoutErrorFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag)

Get timerout error flag.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_timeout_error_flag | Pointer to timeout error flag |

### ◆ patternErrorFlagGet

fsp_err_t(* motor_120_control_api_t::patternErrorFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag)

Get pattern error flag.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_pattern_error_flag | Pointer to pattern error flag |

### ◆ voltageRefGet

fsp_err_t(* motor_120_control_api_t::voltageRefGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref)

Get voltage ref.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_voltage_ref | Pointer to flag voltage ref |

### ◆ parameterUpdate

fsp_err_t(* motor_120_control_api_t::parameterUpdate) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)

Update configuration parameters for the calculation in the motor 120 control module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_120_control_instance_t

| struct motor_120_control_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_120_control_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_120_control_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_120_control_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ motor_120_control_ctrl_t

| typedef void motor_120_control_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

### Enumeration Type Documentation

#### ◆ motor_120_control_event_t

| enum motor_120_control_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_120_CONTROL_EVENT_ADC_FORWARD | Event before motor 120 driver process. |
| MOTOR_120_CONTROL_EVENT_ADC_BACKWARD | Event after motor 120 driver process. |
| MOTOR_120_CONTROL_EVENT_CYCLE_FORWARD | Before cyclic process of speed control. |
| MOTOR_120_CONTROL_EVENT_CYCLE_BACKWARD | After cyclic process of speed control. |

#### ◆ motor_120_conduction_type_t

| enum motor_120_conduction_type_t | |
|---|---|
| Enumerator | |
| MOTOR_120_CONDUCTION_TYPE_FIRST60 | First 60 degree PWM. |
| MOTOR_120_CONDUCTION_TYPE_COMPLEMENTARY | Complementary first 60 degree PWM. |

### ◆ motor_120_control_status_t

| enum motor_120_control_status_t | |
|---|---|
| 120 control status | |
| Enumerator | |
| MOTOR_120_CONTROL_STATUS_INACTIVE | 120 control status inactive |
| MOTOR_120_CONTROL_STATUS_ACTIVE | 120 control status active |

### ◆ motor_120_control_run_mode_t

| enum motor_120_control_run_mode_t | |
|---|---|
| Run mode | |
| Enumerator | |
| MOTOR_120_CONTROL_RUN_MODE_INIT | Run mode init. |
| MOTOR_120_CONTROL_RUN_MODE_BOOT | Run mode boot. |
| MOTOR_120_CONTROL_RUN_MODE_DRIVE | Run mode drive. |

### ◆ motor_120_control_rotation_direction_t

| enum motor_120_control_rotation_direction_t | |
|---|---|
| Rotation direction | |
| Enumerator | |
| MOTOR_120_CONTROL_ROTATION_DIRECTION_CW | Clockwise. |
| MOTOR_120_CONTROL_ROTATION_DIRECTION_CCW | Counter clockwise. |
| MOTOR_120_CONTROL_ROTATION_DIRECTION_MAX | Maximum value. |

### ◆ motor_120_control_wait_stop_flag_t

| enum motor_120_control_wait_stop_flag_t | |
|---|---|
| Flag for waiting for motor stop | |
| Enumerator | |
| MOTOR_120_CONTROL_WAIT_STOP_FLAG_CLEAR | Wait stop flag clear. |
| MOTOR_120_CONTROL_WAIT_STOP_FLAG_SET | Wait stop flag set. |

### ◆ motor_120_control_timeout_error_flag_t

| enum motor_120_control_timeout_error_flag_t | |
|---|---|
| Flag for timeout error status | |
| Enumerator | |
| MOTOR_120_CONTROL_TIMEOUT_ERROR_FLAG_CLEAR | Timeout error flag clear. |
| MOTOR_120_CONTROL_TIMEOUT_ERROR_FLAG_SET | Timeout error flag set. |

### ◆ motor_120_control_pattern_error_flag_t

| enum motor_120_control_pattern_error_flag_t | |
|---|---|
| Flag for pattern error status | |
| Enumerator | |
| MOTOR_120_CONTROL_PATTERN_ERROR_FLAG_CLEAR | Pattern error flag clear. |
| MOTOR_120_CONTROL_PATTERN_ERROR_FLAG_SET | Pattern error flag set. |

### ◆ motor_120_control_speed_ref_t

| enum motor_120_control_speed_ref_t | |
|---|---|
| Speed reference status | |
| Enumerator | |
| MOTOR_120_CONTROL_SPEED_REF_ZERO_CONST | Speed reference zero const. |
| MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_1 | Speed reference openloop 1. |
| MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_2 | Speed reference openloop 2. |
| MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_3 | Speed reference openloop 3. |
| MOTOR_120_CONTROL_SPEED_REF_CHANGE | Speed reference change. |

### ◆ motor_120_control_voltage_ref_t

| enum motor_120_control_voltage_ref_t | |
|---|---|
| Voltage reference status | |
| Enumerator | |
| MOTOR_120_CONTROL_VOLTAGE_REF_ZERO_CONST | Voltage reference zero const. |
| MOTOR_120_CONTROL_VOLTAGE_REF_UP | Voltage reference up. |
| MOTOR_120_CONTROL_VOLTAGE_REF_CONST | Voltage reference const. |
| MOTOR_120_CONTROL_VOLTAGE_REF_OPENLOOP | Voltage reference opneloop. |
| MOTOR_120_CONTROL_VOLTAGE_REF_PI_OUTPUT | Voltage reference pi output. |

## 5.3.10.2 Motor 120-Degree Driver Interface

Interfaces » Motor

### Detailed Description

Interface for motor driver functions.

# Summary

The MOTOR_120_DRIVER interface for setting the PWM modulation duty

### Data Structures

| | |
|---|---|
| struct | motor_120_driver_callback_args_t |
| struct | motor_120_driver_current_status_t |
| struct | motor_120_driver_cfg_t |
| struct | motor_120_driver_api_t |
| struct | motor_120_driver_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_120_driver_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_120_driver_event_t |
| enum | motor_120_driver_flag_offset_calc_t |
| enum | motor_120_driver_phase_pattern_t |

## Data Structure Documentation

### ◆ motor_120_driver_callback_args_t

| struct motor_120_driver_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| motor_120_driver_event_t | event | Event trigger. |
| void * | p_context | Placeholder for user data. |

### ◆ motor_120_driver_current_status_t

| struct motor_120_driver_current_status_t | | |
|---|---|---|
| Current data get structure | | |
| Data Fields | | |
| float | iu | U phase current (A) |
| float | iv | V phase current (A) |
| float | iw | W phase current (A) |
| | | |

| | | |
|---|---|---|
| float | vdc | Main line voltage (V) |
| float | vu | U phase voltage (V) |
| float | vv | V phase voltage (V) |
| float | vw | W phase voltage (V) |

### ◆ motor_120_driver_cfg_t

| struct motor_120_driver_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| void(* | p_callback )(motor_120_driver_callback_args_t *p_args) |
| | Callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

### ◆ motor_120_driver_api_t

| struct motor_120_driver_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_120_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | run )(motor_120_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(motor_120_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_120_driver_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | phaseVoltageSet )(motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage) |
| | |
| fsp_err_t(* | phasePatternSet )(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern) |
| | |
| fsp_err_t(* | currentGet )(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status) |
| | |
| fsp_err_t(* | currentOffsetCalc )(motor_120_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | flagCurrentOffsetGet )(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_120_driver_api_t::open) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)

Initialize the motor 120 driver module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_120_driver_api_t::close) (motor_120_driver_ctrl_t *const p_ctrl)

Close the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ run

fsp_err_t(* motor_120_driver_api_t::run) (motor_120_driver_ctrl_t *const p_ctrl)

Run the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stop

fsp_err_t(* motor_120_driver_api_t::stop) (motor_120_driver_ctrl_t *const p_ctrl)

Stop the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_120_driver_api_t::reset) (motor_120_driver_ctrl_t *const p_ctrl)

Reset variables of the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ phaseVoltageSet

fsp_err_t(* motor_120_driver_api_t::phaseVoltageSet) (motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

Set (Input) phase voltage data into the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | u_voltage | U phase voltage [V] |
| [in] | v_voltage | V phase voltage [V] |
| [in] | w_voltage | W phase voltage [V] |

### ◆ phasePatternSet

fsp_err_t(* motor_120_driver_api_t::phasePatternSet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern)

Set phase voltage pattern the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pattern | Voltage pattern |

### ◆ currentGet

fsp_err_t(* motor_120_driver_api_t::currentGet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)

Get phase current, Vdc and Va_max data from the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_current_status | Pointer to get data structure. |

### ◆ currentOffsetCalc

fsp_err_t(* motor_120_driver_api_t::currentOffsetCalc) (motor_120_driver_ctrl_t *const p_ctrl)

current offset detection from the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ flagCurrentOffsetGet

fsp_err_t(* motor_120_driver_api_t::flagCurrentOffsetGet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset)

Get the flag of finish current offset detection from the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_flag_offset | Flag of finish current offset detection |

### ◆ parameterUpdate

| fsp_err_t(* motor_120_driver_api_t::parameterUpdate) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg) |
|---|

Update configuration parameters for the calculation in the motor 120 driver module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_120_driver_instance_t

| struct motor_120_driver_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_120_driver_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_120_driver_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_120_driver_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_120_driver_ctrl_t

| typedef void motor_120_driver_ctrl_t |
|---|

Control block. Allocate an instance specific control block to pass into the API calls.

## Enumeration Type Documentation

### ◆ motor_120_driver_event_t

| enum motor_120_driver_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_120_DRIVER_EVENT_FORWARD | Event before motor 120 driver process (before current control timing) |
| MOTOR_120_DRIVER_EVENT_120_CONTROL | Event 120 detection. |
| MOTOR_120_DRIVER_EVENT_BACKWARD | Event after motor 120 driver process (after PWM duty setting) |

### ◆ motor_120_driver_flag_offset_calc_t

| enum motor_120_driver_flag_offset_calc_t | |
|---|---|
| The flag represents that the offset measurement is finished | |
| Enumerator | |
| MOTOR_120_DRIVER_FLAG_OFFSET_CALC_CLEAR | Offset calculation not finished. |
| MOTOR_120_DRIVER_FLAG_OFFSET_CALC_OFF_FINISH | Off voltage offset calculation finished. |
| MOTOR_120_DRIVER_FLAG_OFFSET_CALC_ALL_FINISH | All offset calculation finished. |

### ◆ motor_120_driver_phase_pattern_t

| enum motor_120_driver_phase_pattern_t | |
|---|---|
| Phase voltage pattern | |
| Enumerator | |
| MOTOR_120_DRIVER_PHASE_PATTERN_ERROR | Phase voltage pattern error. |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_PWM_VN_ON | Up(PWM) to Vn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_PWM_WN_ON | Up(PWM) to Wn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_PWM_UN_ON | Vp(PWM) to Un(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_PWM_WN_ON | Vp(PWM) to Wn(on) |
| | |

| | |
|---|---|
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_PWM_UN_ON | Wp(PWM) to Un(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_PWM_VN_ON | Wp(PWM) to Vn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_VN_PWM | Up(on) to Vn(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_WN_PWM | Up(on) to Wn(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_UN_PWM | Vp(on) to Un(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_WN_PWM | Vp(on) to Wn(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_UN_PWM | Wp(on) to Un(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_VN_PWM | Wp(on) to Vn(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_U_PWM_VN_ON | U(PWM) to Vn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_U_PWM_WN_ON | U(PWM) to Wn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_V_PWM_UN_ON | V(PWM) to Un(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_V_PWM_WN_ON | V(PWM) to Wn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_W_PWM_UN_ON | W(PWM) to Un(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_W_PWM_VN_ON | W(PWM) to Vn(on) |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_V_PWM | Up(on) to V(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_W_PWM | Up(on) to W(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_U_PWM | Vp(on) to U(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_W_PWM | Vp(on) to W(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_U_PWM | Wp(on) to U(PWM) |
| MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_V_PWM | Wp(on) to V(PWM) |

### 5.3.10.3 Motor Inertia Estimate Interface

Interfaces » Motor

#### Detailed Description

Interface for Motor inertia estimate functions.

# Summary

The Motor interface provides Motor inertia estimate functionality.

#### Data Structures

| | |
|---|---|
| struct | motor_inertia_estimate_info_t |
| struct | motor_inertia_estimate_cfg_t |
| struct | motor_inertia_estimate_api_t |
| struct | motor_inertia_estimate_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | motor_inertia_estimate_ctrl_t |

#### Data Structure Documentation

◆ **motor_inertia_estimate_info_t**

| struct motor_inertia_estimate_info_t | | |
|---|---|---|
| Interface data structure | | |
| Data Fields | | |
| int16_t | s2_position_reference_degree | Position reference [degree]. |
| motor_inertia_estimate_mode_t | mode | Internal mode of inertia estimation. |
| float | f_estimated_inertia | Estimated inertia data. |

◆ **motor_inertia_estimate_cfg_t**

| struct motor_inertia_estimate_cfg_t | | |
|---|---|---|
| Configuration parameters. | | |
| Data Fields | | |
| void * | p_context | |
| void const * | p_extend | Placeholder for user extension. |

### ◆ motor_inertia_estimate_api_t

struct motor_inertia_estimate_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | start )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info) |
| | |
| fsp_err_t(* | dataSet )(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data) |
| | |
| fsp_err_t(* | speedCyclic )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | currentCyclic )(motor_inertia_estimate_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_inertia_estimate_api_t::open) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_inertia_estimate_api_t::close) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ start

fsp_err_t(* motor_inertia_estimate_api_t::start) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Start the function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stop

fsp_err_t(* motor_inertia_estimate_api_t::stop) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Stop( same as cancel ) the function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_inertia_estimate_api_t::reset) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Reset the function. (recover from error state)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ infoGet

fsp_err_t(* motor_inertia_estimate_api_t::infoGet) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_info_t *const p_info)

Get information from the function (to set speed & position control)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Pointer to information |

### ◆ dataSet

fsp_err_t(* motor_inertia_estimate_api_t::dataSet) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_set_data_t *const p_set_data)

Set the data to the function (from speed, position and current control)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_set_data | Pointer to set the data |

### ◆ speedCyclic

fsp_err_t(* motor_inertia_estimate_api_t::speedCyclic) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Speed cyclic process of the function

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ currentCyclic

fsp_err_t(* motor_inertia_estimate_api_t::currentCyclic) (motor_inertia_estimate_ctrl_t *const p_ctrl)

Current cyclic process of the function

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ parameterUpdate

| fsp_err_t(* motor_inertia_estimate_api_t::parameterUpdate) (motor_inertia_estimate_ctrl_t *const p_ctrl, motor_inertia_estimate_cfg_t const *p_cfg) |
|---|

Update parameters for the function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_inertia_estimate_instance_t

| struct motor_inertia_estimate_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_inertia_estimate_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_inertia_estimate_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_inertia_estimate_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_inertia_estimate_ctrl_t

| typedef void motor_inertia_estimate_ctrl_t |
|---|
| Motor inertia estimate block. Allocate an instance specific control block to pass into the API calls. |

### 5.3.10.4 Motor Interface

Interfaces » Motor

## Detailed Description

Interface for Motor functions.

# Summary

The Motor interface provides Motor functionality.

### Data Structures

| | |
|---|---|
| struct | motor_callback_args_t |
| struct | motor_cfg_t |
| struct | motor_api_t |
| struct | motor_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_error_t |
| enum | motor_callback_event_t |
| enum | motor_wait_stop_flag_t |
| enum | motor_function_select_t |

## Data Structure Documentation

### ◆ motor_callback_args_t

| struct motor_callback_args_t | | |
|---|---|---|
| callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| motor_callback_event_t | event | |

### ◆ motor_cfg_t

| struct motor_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| motor_speed_instance_t const * | p_motor_speed_instance |
| | Speed Instance. |
| | |
| motor_current_instance_t const * | p_motor_current_instance |
| | |

| | |
|---|---|
| | Current Instance. |
| | |
| void(* | p_callback )(motor_callback_args_t *p_args) |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ p_callback

| |
|---|
| void(* motor_cfg_t::p_callback) (motor_callback_args_t *p_args) |
| Placeholder for user data. Passed to the user callback in motor_callback_args_t. |

### ◆ motor_api_t

| |
|---|
| struct motor_api_t |
| Functions implemented at the HAL layer will follow this API. |
| **Data Fields** |

| | |
|---|---|
| fsp_err_t(* | open )(motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | run )(motor_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(motor_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | errorSet )(motor_ctrl_t *const p_ctrl, motor_error_t const error) |
| | |
| fsp_err_t(* | speedSet )(motor_ctrl_t *const p_ctrl, float const speed_rpm) |
| | |
| fsp_err_t(* | positionSet )(motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) |
| | |

| | |
|---|---|
| fsp_err_t(* | statusGet )(motor_ctrl_t *const p_ctrl, uint8_t *const p_status) |
| | |
| fsp_err_t(* | angleGet )(motor_ctrl_t *const p_ctrl, float *const p_angle_rad) |
| | |
| fsp_err_t(* | speedGet )(motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) |
| | |
| fsp_err_t(* | waitStopFlagGet )(motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) |
| | |
| fsp_err_t(* | errorCheck )(motor_ctrl_t *const p_ctrl, uint16_t *const p_error) |
| | |
| fsp_err_t(* | functionSelect )(motor_ctrl_t *const p_ctrl, motor_function_select_t const function) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_api_t::open) (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_api_t::close) (motor_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ run

fsp_err_t(* motor_api_t::run) (motor_ctrl_t *const p_ctrl)

Run the motor. (Start the motor rotation.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stop

fsp_err_t(* motor_api_t::stop) (motor_ctrl_t *const p_ctrl)

Stop the motor. (Stop the motor rotation.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_api_t::reset) (motor_ctrl_t *const p_ctrl)

Reset the motor control. (Recover from the error status.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ errorSet

fsp_err_t(* motor_api_t::errorSet) (motor_ctrl_t *const p_ctrl, motor_error_t const error)

Set Error Information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | error | Happened error code |

### ◆ speedSet

fsp_err_t(* motor_api_t::speedSet) (motor_ctrl_t *const p_ctrl, float const speed_rpm)

Set rotation speed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | speed_rpm | Required rotation speed [rpm] |

### ◆ positionSet

fsp_err_t(* motor_api_t::positionSet) (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)

Set reference position.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_position | Pointer to set required data |

### ◆ statusGet

fsp_err_t(* motor_api_t::statusGet) (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)

Get the motor control status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_status | Pointer to get the motor control status |

### ◆ angleGet

fsp_err_t(* motor_api_t::angleGet) (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)

Get the rotor angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_angle_rad | Pointer to get the rotor angle [rad] |

### ◆ speedGet

fsp_err_t(* motor_api_t::speedGet) (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)

Get the rotation speed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_speed_rpm | Pointer to get the rotation speed [rpm] |

### ◆ waitStopFlagGet

| fsp_err_t(* motor_api_t::waitStopFlagGet) (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) |
|---|

Get wait stop flag.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_flag | Pointer to wait stop flag |

### ◆ errorCheck

| fsp_err_t(* motor_api_t::errorCheck) (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) |
|---|

Check the error occurrence

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_error | Pointer to get occurred error |

### ◆ functionSelect

| fsp_err_t(* motor_api_t::functionSelect) (motor_ctrl_t *const p_ctrl, motor_function_select_t const function) |
|---|

FunctionSelect.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | function | Selected function |

### ◆ motor_instance_t

| struct motor_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_ctrl_t

| typedef void motor_ctrl_t |
| --- |
| Motor Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ motor_error_t

| enum motor_error_t |
| --- |
| Error information |

### ◆ motor_callback_event_t

| enum motor_callback_event_t | |
| --- | --- |
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_CALLBACK_EVENT_SPEED_FORWARD | Event forward speed control. |
| MOTOR_CALLBACK_EVENT_SPEED_BACKWARD | Event backward speed control. |
| MOTOR_CALLBACK_EVENT_CURRENT_FORWARD | Event forward current control. |
| MOTOR_CALLBACK_EVENT_CURRENT_BACKWARD | Event backward current control. |
| MOTOR_CALLBACK_EVENT_ADC_FORWARD | Event before motor 120 driver process. |
| MOTOR_CALLBACK_EVENT_ADC_BACKWARD | Event after motor 120 driver process. |
| MOTOR_CALLBACK_EVENT_CYCLE_FORWARD | Before cyclic process of speed control. |
| MOTOR_CALLBACK_EVENT_CYCLE_BACKWARD | After cyclic process of speed control. |

### ◆ motor_wait_stop_flag_t

| enum motor_wait_stop_flag_t | |
|---|---|
| Flag for waiting for motor stop | |
| Enumerator | |
| MOTOR_WAIT_STOP_FLAG_CLEAR | Wait stop flag clear. |
| MOTOR_WAIT_STOP_FLAG_SET | Wait stop flag set. |

### ◆ motor_function_select_t

| enum motor_function_select_t | |
|---|---|
| Function select | |
| Enumerator | |
| MOTOR_FUNCTION_SELECT_NONE | No function selected. |
| MOTOR_FUNCTION_SELECT_INERTIA_ESTIMATE | Inertia estimation. |
| MOTOR_FUNCTION_SELECT_RETURN_ORIGIN | Return origin position. |

## 5.3.10.5 Motor Return Origin Function Interface

Interfaces » Motor

### Detailed Description

Interface for Motor return origin functions.

# Summary

The Motor interface provides Motor return origin functionality.

### Data Structures

| | |
|---|---|
| struct | motor_return_origin_info_t |
| struct | motor_return_origin_cfg_t |
| struct | motor_return_origin_api_t |

| | |
|---|---|
| struct | motor_return_origin_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_return_origin_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_return_origin_mode_t |

## Data Structure Documentation

### ◆ motor_return_origin_info_t

| struct motor_return_origin_info_t | | |
|---|---|---|
| Interface data structure | | |
| Data Fields | | |
| float | f_position_reference_degree | Position reference [degree]. |
| motor_return_origin_state_t | state | |
| float | f_result_angle | Result angle position. |

### ◆ motor_return_origin_cfg_t

| struct motor_return_origin_cfg_t | | |
|---|---|---|
| Configuration parameters. | | |
| Data Fields | | |
| motor_return_origin_mode_t | mode | |
| void * | p_context | |
| void const * | p_extend | Placeholder for user extension. |

### ◆ motor_return_origin_api_t

| struct motor_return_origin_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_return_origin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | start )(motor_return_origin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(motor_return_origin_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | |
| fsp_err_t(* | reset )(motor_return_origin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | infoGet )(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info) |
| | |
| fsp_err_t(* | dataSet )(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data) |
| | |
| fsp_err_t(* | speedCyclic )(motor_return_origin_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_return_origin_api_t::open) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_return_origin_api_t::close) (motor_return_origin_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ start

fsp_err_t(* motor_return_origin_api_t::start) (motor_return_origin_ctrl_t *const p_ctrl)

Start the function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stop

fsp_err_t(* motor_return_origin_api_t::stop) (motor_return_origin_ctrl_t *const p_ctrl)

Stop the function. (Cancel the function works.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_return_origin_api_t::reset) (motor_return_origin_ctrl_t *const p_ctrl)

Reset the function. (Initialize the function.)

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ infoGet

fsp_err_t(* motor_return_origin_api_t::infoGet) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_info_t *const p_info)

Get the function information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Pointer to info |

### ◆ dataSet

fsp_err_t(* motor_return_origin_api_t::dataSet) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_set_data_t *const p_set_data)

Set the data to the function

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_set_data | Pointer to set the data |

### ◆ speedCyclic

fsp_err_t(* motor_return_origin_api_t::speedCyclic) (motor_return_origin_ctrl_t *const p_ctrl)

Speed cyclic process of the function

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ parameterUpdate

fsp_err_t(* motor_return_origin_api_t::parameterUpdate) (motor_return_origin_ctrl_t *const p_ctrl, motor_return_origin_cfg_t const *p_cfg)

Update parameters for the function.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_return_origin_instance_t

struct motor_return_origin_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| motor_return_origin_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_return_origin_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_return_origin_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_return_origin_ctrl_t

| typedef void motor_return_origin_ctrl_t |
|---|
| Motor return origin function block. Allocate an instance specific control block to pass into the API calls. |

#### Enumeration Type Documentation

### ◆ motor_return_origin_mode_t

| enum motor_return_origin_mode_t | |
|---|---|
| Selection type of return origin function | |
| Enumerator | |
| MOTOR_RETURN_ORIGIN_MODE_PUSH | Return origin position with pushing. |
| MOTOR_RETURN_ORIGIN_MODE_SENSOR | Return origin position with origin sensor. |
| MOTOR_RETURN_ORIGIN_MODE_2_SENSOR | Return origin position with 2 sensors. |
| MOTOR_RETURN_ORIGIN_MODE_3_SENSOR | Return origin position with 3 sensors. |

## 5.3.10.6 Motor angle Interface

Interfaces » Motor

#### Detailed Description

Interface for motor angle and speed calculation functions.

# Summary

The Motor angle interface calculates the rotor angle and rotational speed from other data.

#### Data Structures

| | |
|---|---|
| struct | motor_angle_cfg_t |
| struct | motor_angle_current_t |
| struct | motor_angle_voltage_reference_t |
| struct | motor_angle_ad_data_t |

| | |
|---|---|
| struct | motor_angle_encoder_info_t |
| struct | motor_angle_api_t |
| struct | motor_angle_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_angle_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_sense_encoder_angle_adjust_t |
| enum | motor_angle_open_loop_t |
| enum | motor_angle_error_t |

## Data Structure Documentation

### ◆ motor_angle_cfg_t

| struct motor_angle_cfg_t |
|---|
| Configuration parameters. |

### ◆ motor_angle_current_t

| struct motor_angle_current_t | | |
|---|---|---|
| Interface structure | | |
| Data Fields | | |
| float | id | d-axis current |
| float | iq | q-axis current |

### ◆ motor_angle_voltage_reference_t

| struct motor_angle_voltage_reference_t | | |
|---|---|---|
| Motor angle voltage reference | | |
| Data Fields | | |
| float | vd | d-axis voltage reference |
| float | vq | q-axis voltage reference |

### ◆ motor_angle_ad_data_t

| struct motor_angle_ad_data_t |
|---|
| A/D conversion data |
| Data Fields |

| | | |
|---|---|---|
| float | sin_ad_data | sin A/D data of induction sensor |
| float | cos_ad_data | cos A/D data of induction sensor |

### ◆ motor_angle_encoder_info_t

| struct motor_angle_encoder_info_t | | |
|---|---|---|
| Motor angle encoder adjustment info | | |
| Data Fields | | |
| motor_sense_encoder_angle_adjust_t | e_adjust_status | Encoder Adjustment Status. |
| uint8_t | u1_adjust_count_full | Adjustment count became full. |
| uint8_t | u1_adjust_mode | Adjustment mode (only valid with ENCODER) |
| motor_angle_open_loop_t | e_open_loop_status | Openloop status. |
| float | f_openloop_speed | Openloop speed. |
| float | f_openloop_id_ref | Openloop d-axis current. |

### ◆ motor_angle_api_t

| struct motor_angle_api_t | |
|---|---|
| Functions implemented as application interface will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | currentSet )(motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage) |
| | |
| fsp_err_t(* | speedSet )(motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed) |
| | |
| fsp_err_t(* | flagPiCtrlSet )(motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) |
| | |

| | |
|---|---|
| fsp_err_t(* | internalCalculate )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | angleSpeedGet )(motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err) |
| | |
| fsp_err_t(* | angleAdjust )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | encoderCyclic )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | cyclicProcess )(motor_angle_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | sensorDataSet )(motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data) |
| | |
| fsp_err_t(* | estimatedComponentGet )(motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq) |
| | |
| fsp_err_t(* | infoGet )(motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_angle_api_t::open) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)

Initialize the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_angle_api_t::close) (motor_angle_ctrl_t *const p_ctrl)

Close (Finish) the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_angle_api_t::reset) (motor_angle_ctrl_t *const p_ctrl)

Reset the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ currentSet

fsp_err_t(* motor_angle_api_t::currentSet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)

Set (Input) Current & Voltage Reference data into the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_st_current | Pointer to current structure |
| [in] | p_st_voltage | Pointer to voltage Reference structure |

### ◆ speedSet

fsp_err_t(* motor_angle_api_t::speedSet) (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)

Set (Input) Speed Information into the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | speed_ctrl | Control reference of rotational speed [rad/s] |
| [in] | damp_speed | Damping rotational speed [rad/s] |

### ◆ flagPiCtrlSet

fsp_err_t(* motor_angle_api_t::flagPiCtrlSet) (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)

Set the flag of PI Control runs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | flag_pi | The flag of PI control runs |

### ◆ internalCalculate

fsp_err_t(* motor_angle_api_t::internalCalculate) (motor_angle_ctrl_t *const p_ctrl)

Calculate internal parameters of encoder process.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ angleSpeedGet

fsp_err_t(* motor_angle_api_t::angleSpeedGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)

Get rotor angle and rotational speed from the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_angl | Memory address to get rotor angle data |
| [out] | p_speed | Memory address to get rotational speed data |
| [out] | p_phase_err | Memory address to get phase(angle) error data |

### ◆ angleAdjust

fsp_err_t(* motor_angle_api_t::angleAdjust) (motor_angle_ctrl_t *const p_ctrl)

Angle Adjustment Process.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ encoderCyclic

fsp_err_t(* motor_angle_api_t::encoderCyclic) (motor_angle_ctrl_t *const p_ctrl)

DEPRECATED Encoder Cyclic Process.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ cyclicProcess

fsp_err_t(* motor_angle_api_t::cyclicProcess) (motor_angle_ctrl_t *const p_ctrl)

Cyclic Process. please

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ sensorDataSet

fsp_err_t(* motor_angle_api_t::sensorDataSet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_ad_data_t *const p_ad_data)

Set sensor A/D data into the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_ad_data | Pointer to A/D conversion data |

### ◆ estimatedComponentGet

fsp_err_t(* motor_angle_api_t::estimatedComponentGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)

Get estimated d/q-axis component from the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_ed | Memory address to get estimated d-axis component |
| [out] | p_eq | Memory address to get estimated q-axis component |

### ◆ infoGet

| fsp_err_t(* motor_angle_api_t::infoGet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info) |
|---|

Get Encoder Calculate Information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Memory address to get angle internal information |

### ◆ parameterUpdate

| fsp_err_t(* motor_angle_api_t::parameterUpdate) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg) |
|---|

Update Parameters for the calculation in the Motor_Angle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_angle_instance_t

| struct motor_angle_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_angle_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_angle_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_angle_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_angle_ctrl_t

| typedef void motor_angle_ctrl_t |
|---|
| Motor Angle Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ motor_sense_encoder_angle_adjust_t

| enum motor_sense_encoder_angle_adjust_t | |
|---|---|
| Enumerator | |
| MOTOR_SENSE_ENCODER_ANGLE_ADJUST_90_DEGREE | Roter Angle Adjustment to pull in 90degree. |
| MOTOR_SENSE_ENCODER_ANGLE_ADJUST_0_DEGREE | Roter Angle Adjustment to pull in 0degree. |
| MOTOR_SENSE_ENCODER_ANGLE_ADJUST_FINISH | Roter Angle Adjustment Finish. |
| MOTOR_SENSE_ENCODER_ANGLE_ADJUST_OPENLOOP | Roter Angle Adjustment Finish. |

### ◆ motor_angle_open_loop_t

| enum motor_angle_open_loop_t | |
|---|---|
| Enumerator | |
| MOTOR_ANGLE_OPEN_LOOP_INACTIVE | Openloop inactive. |
| MOTOR_ANGLE_OPEN_LOOP_ACTIVE | Openloop active. |

### ◆ motor_angle_error_t

| enum motor_angle_error_t | |
|---|---|
| Flag for induction correction error status | |
| Enumerator | |
| MOTOR_ANGLE_ERROR_NONE | No error happen. |
| MOTOR_ANGLE_ERROR_INDUCTION | Error happens in induction sensor correction process. |

## 5.3.10.7 Motor current Interface

Interfaces » Motor

### Detailed Description

Interface for motor current functions.

# Summary

The Motor current interface for getting the PWM modulation duty from electric current and speed

### Data Structures

| | |
|---|---|
| struct | motor_current_output_t |
| struct | motor_current_input_current_t |
| struct | motor_current_input_voltage_t |
| struct | motor_current_get_voltage_t |
| struct | motor_current_cfg_t |
| struct | motor_current_api_t |
| struct | motor_current_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_current_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_current_event_t |

## Data Structure Documentation

### ◆ motor_current_output_t

| struct motor_current_output_t | | |
|---|---|---|
| Structure of interface to speed control Output parameters | | |
| Data Fields | | |
| float | f_id | D-axis current [A]. |
| float | f_iq | Q-axis current [A]. |
| float | f_vamax | |
| float | f_speed_rad | Speed value [rad/s]. |
| float | f_speed_rpm | Speed value [rpm]. |
| float | f_rotor_angle | Motor rotor angle [rad]. |
| float | f_position_rad | Motor rotor position [rad]. |
| float | f_ed | Estimated d-axis component[V] of flux due to the permanent magnet. |
| float | f_eq | Estimated q-axis component[V] |

| | | |
|---|---|---|
| | | of flux due to the permanent magnet. |
| float | f_phase_err_rad | Phase error [rad]. |
| uint8_t | u1_flag_get_iref | Flag to set d/q-axis current reference. |
| uint8_t | u1_adjust_status | Angle adjustment satatus. |
| uint8_t | u1_adjust_count_full | Angle adjustment count full. |
| uint8_t | u1_adjust_mode | Angle adjustment mode (only valid with ENCODER) |
| uint8_t | u1_openloop_status | Openloop status. |
| float | f_openloop_speed | Openloop speed. |
| float | f_openloop_id_ref | Openloop d-axis current. |

### ◆ motor_current_input_current_t

| struct motor_current_input_current_t | | |
|---|---|---|
| Three-phase input current | | |
| Data Fields | | |
| float | iu | U phase current[A]. |
| float | iv | V phase current[A]. |
| float | iw | W phase current[A]. |

### ◆ motor_current_input_voltage_t

| struct motor_current_input_voltage_t | | |
|---|---|---|
| Input voltage | | |
| Data Fields | | |
| float | vdc | Main line voltage[V]. |
| float | va_max | Maximum magnitude of voltage vector[V]. |

### ◆ motor_current_get_voltage_t

| struct motor_current_get_voltage_t | | |
|---|---|---|
| Struct to get motor current | | |
| Data Fields | | |
| float | u_voltage | U phase voltage[V]. |
| float | v_voltage | V phase voltage[V]. |
| float | w_voltage | W phase voltage[V]. |
| float | vd_reference | d-axis voltage reference |
| float | vq_reference | q-axis voltage reference |

### ◆ motor_current_cfg_t

| struct motor_current_cfg_t |
| --- |
| Configuration parameters. |

### ◆ motor_current_api_t

| struct motor_current_api_t | |
| --- | --- |
| Functions implemented at the Motor Current Module will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_current_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_current_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | run )(motor_current_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | parameterSet )(motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input) |
| | |
| fsp_err_t(* | currentReferenceSet )(motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference) |
| | |
| fsp_err_t(* | speedPhaseSet )(motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad) |
| | |
| fsp_err_t(* | currentSet )(motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage) |
| | |
| fsp_err_t(* | parameterGet )(motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output) |
| | |
| fsp_err_t(* | currentGet )(motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq) |
| | |
| fsp_err_t(* | phaseVoltageGet )(motor_current_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | motor_current_get_voltage_t *const p_voltage) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_current_api_t::open) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Initialize the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_current_api_t::close) (motor_current_ctrl_t *const p_ctrl)

Close (Finish) the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_current_api_t::reset) (motor_current_ctrl_t *const p_ctrl)

Reset variables for the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ run

fsp_err_t(* motor_current_api_t::run) (motor_current_ctrl_t *const p_ctrl)

Activate the motor current control.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ parameterSet

fsp_err_t(* motor_current_api_t::parameterSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)

Set (Input) parameters into the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_st_input | Pointer to input data structure(speed control output data) |

### ◆ currentReferenceSet

fsp_err_t(* motor_current_api_t::currentReferenceSet) (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)

Set (Input) Current reference into the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | id_reference | D-axis current reference [A] |
| [in] | iq_reference | Q-axis current reference [A] |

### ◆ speedPhaseSet

fsp_err_t(* motor_current_api_t::speedPhaseSet) (motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad)

Set (Input) Speed & Phase data into the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | speed_rad | Rotational speed [rad/s] |
| [in] | phase_rad | Rotor phase [rad] |

### ◆ currentSet

fsp_err_t(* motor_current_api_t::currentSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)

Set (Input) Current data into the motor current module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_st_current | Pointer to input current structure |
| [in] | p_st_voltage | Pointer to input voltage structure |

### ◆ parameterGet

fsp_err_t(* motor_current_api_t::parameterGet) (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)

Get (output) parameters from the motor current module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_st_output | Pointer to output data structure(speed control input data) |

### ◆ currentGet

fsp_err_t(* motor_current_api_t::currentGet) (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)

Get d/q-axis current

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_id | Pointer to get d-axis current [A] |
| [out] | p_iq | Pointer to get q-axis current [A] |

### ◆ phaseVoltageGet

| fsp_err_t(* motor_current_api_t::phaseVoltageGet) (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage) |
|---|

Get phase output voltage

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_voltage | Pointer to get voltages |

### ◆ parameterUpdate

| fsp_err_t(* motor_current_api_t::parameterUpdate) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) |
|---|

Update parameters for the calculation in the motor current control.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_current_instance_t

| struct motor_current_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_current_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_current_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_current_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_current_ctrl_t

| typedef void motor_current_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ motor_current_event_t

| enum motor_current_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_CURRENT_EVENT_FORWARD | Event forward current control. |
| MOTOR_CURRENT_EVENT_DATA_SET | Event set speed control output data. |
| MOTOR_CURRENT_EVENT_BACKWARD | Event backward current control. |

## 5.3.10.8 Motor driver Interface

Interfaces » Motor

### Detailed Description

Interface for motor driver functions.

# Summary

The Motor driver interface for setting the PWM modulation duty

### Data Structures

| | |
|---|---|
| struct | motor_driver_callback_args_t |
| struct | motor_driver_current_get_t |
| struct | motor_driver_cfg_t |
| struct | motor_driver_api_t |
| struct | motor_driver_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_driver_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_driver_event_t |
| enum | motor_driver_shunt_type_t |

## Data Structure Documentation

### ◆ motor_driver_callback_args_t

| struct motor_driver_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| motor_driver_event_t | event | Event trigger. |
| void * | p_context | Placeholder for user data. |

### ◆ motor_driver_current_get_t

| struct motor_driver_current_get_t | | |
|---|---|---|
| Current Data Get Structure | | |
| Data Fields | | |
| float | iu | U phase current [A]. |
| float | iv | V phase current [A]. |
| float | iw | W phase current [A]. |
| float | vdc | Main Line Voltage [V]. |
| float | va_max | maximum magnitude of voltage vector |
| float | sin_ad | Induction sensor sin signal. |
| float | cos_ad | Induction sensor cos signal. |

### ◆ motor_driver_cfg_t

| struct motor_driver_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| adc_channel_t | iu_ad_ch |
| | A/D Channel for U Phase Current. |
| | |
| adc_channel_t | iv_ad_ch |
| | A/D Channel for V Phase Current. |
| | |
| adc_channel_t | iw_ad_ch |
| | A/D Channel for W Phase Current. |

| | |
|---|---|
| | |
| adc_channel_t | vdc_ad_ch |
| | A/D Channel for Main Line Voltage. |
| | |
| adc_channel_t | sin_ad_ch |
| | A/D Channel for induction sensor sin signal. |
| | |
| adc_channel_t | cos_ad_ch |
| | A/D Channel for induction sensor cos signal. |
| | |
| motor_driver_shunt_type_t | shunt |
| | Selection of shunt type. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |

### ◆ motor_driver_api_t

| struct motor_driver_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | phaseVoltageSet )(motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage) |
| | |
| fsp_err_t(* | currentGet )(motor_driver_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | motor_driver_current_get_t *const p_current_get) |
| | |
| fsp_err_t(* | flagCurrentOffsetGet )(motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset) |
| | |
| fsp_err_t(* | currentOffsetRestart )(motor_driver_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_driver_api_t::open) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Initialize the Motor Driver Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_driver_api_t::close) (motor_driver_ctrl_t *const p_ctrl)

Close the Motor Driver Module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_driver_api_t::reset) (motor_driver_ctrl_t *const p_ctrl)

Reset variables of the Motor Driver Module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ phaseVoltageSet

fsp_err_t(* motor_driver_api_t::phaseVoltageSet) (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

Set (Input) Phase Voltage data into the Motor Driver Module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | u_voltage | U phase voltage [V] |
| [in] | v_voltage | V phase voltage [V] |
| [in] | w_voltage | W phase voltage [V] |

### ◆ currentGet

fsp_err_t(* motor_driver_api_t::currentGet) (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get)

Get Phase current, Vdc and Va_max data from the Motor Driver Module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_current_get | Pointer to get data structure. |

### ◆ flagCurrentOffsetGet

fsp_err_t(* motor_driver_api_t::flagCurrentOffsetGet) (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)

Get the flag of finish current offset detection from the Motor Driver Module

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_flag_offset | Flag of finish current offset detection |

### ◆ currentOffsetRestart

fsp_err_t(* motor_driver_api_t::currentOffsetRestart) (motor_driver_ctrl_t *const p_ctrl)

Restart current offset detection

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ parameterUpdate

| fsp_err_t(* motor_driver_api_t::parameterUpdate) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) |
|---|
| Update Configuration Parameters for the calculation in the Motor Driver Module |

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_driver_instance_t

| struct motor_driver_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_driver_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_driver_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_driver_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_driver_ctrl_t

| typedef void motor_driver_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ motor_driver_event_t

| enum motor_driver_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_DRIVER_EVENT_FORWARD | Event before Motor Driver Process (before Current Control timing) |
| MOTOR_DRIVER_EVENT_CURRENT | Event Current Control timing. |
| MOTOR_DRIVER_EVENT_BACKWARD | Event after Motor Driver Process (after PWM duty setting) |

### ◆ motor_driver_shunt_type_t

| enum motor_driver_shunt_type_t | |
|---|---|
| Selection of shunt type | |
| Enumerator | |
| MOTOR_DRIVER_SHUNT_TYPE_1_SHUNT | Only use U phase current. |
| MOTOR_DRIVER_SHUNT_TYPE_2_SHUNT | Use U and W phase current. |
| MOTOR_DRIVER_SHUNT_TYPE_3_SHUNT | Use all phase current. |

## 5.3.10.9 Motor position Interface

Interfaces » Motor

### Detailed Description

Interface for motor position functions.

# Summary

The Motor position interface for getting the speed references from Encoder Sensor

### Data Structures

| | |
|---|---|
| struct | motor_position_info_t |
| struct | motor_position_cfg_t |

| | |
|---|---|
| struct | motor_position_api_t |
| struct | motor_position_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_position_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | motor_position_ctrl_mode_t |

## Data Structure Documentation

### ◆ motor_position_info_t

| struct motor_position_info_t | | |
|---|---|---|
| Position information | | |
| Data Fields | | |
| uint8_t | u1_state_position_profile | Position control profile state. |
| int16_t | s2_position_degree | Position data [degree]. |

### ◆ motor_position_cfg_t

| struct motor_position_cfg_t | | |
|---|---|---|
| Configuration parameters. | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| void const * | p_extend | |

### ◆ motor_position_api_t

| struct motor_position_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_position_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_position_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | positionGet )(motor_position_ctrl_t *const p_ctrl, int16_t *const p_position) |

| | |
|---|---|
| fsp_err_t(* | positionSet )(motor_position_ctrl_t *const p_ctrl, float const position_rad) |
| fsp_err_t(* | positionReferenceSet )(motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg) |
| fsp_err_t(* | controlModeSet )(motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode) |
| fsp_err_t(* | positionControl )(motor_position_ctrl_t *const p_ctrl) |
| fsp_err_t(* | ipdSpeedPControl )(motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref) |
| fsp_err_t(* | speedReferencePControlGet )(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) |
| fsp_err_t(* | speedReferenceIpdControlGet )(motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref) |
| fsp_err_t(* | speedReferenceFeedforwardGet )(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) |
| fsp_err_t(* | infoGet )(motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info) |
| fsp_err_t(* | parameterUpdate )(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) |

## Field Documentation

### ◆ open

fsp_err_t(* motor_position_api_t::open) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Initialize the Motor Position Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_position_api_t::close) (motor_position_ctrl_t *const p_ctrl)

Close (Finish) the Motor Position Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_position_api_t::reset) (motor_position_ctrl_t *const p_ctrl)

Reset(Stop) the Motor Position Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ positionGet

fsp_err_t(* motor_position_api_t::positionGet) (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)

Get Position data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_position | Pointer to get position data |

### ◆ positionSet

fsp_err_t(* motor_position_api_t::positionSet) (motor_position_ctrl_t *const p_ctrl, float const position_rad)

Set Position data from Encoder.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | position_rad | Position data [radian] |

### ◆ positionReferenceSet

fsp_err_t(* motor_position_api_t::positionReferenceSet) (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)

Set (Input) Position reference into the Motor Position Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | position_reference_deg | Position reference [degree] |

### ◆ controlModeSet

fsp_err_t(* motor_position_api_t::controlModeSet) (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)

Set (Input) Position Control Mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | mode | Position Control Mode |

### ◆ positionControl

fsp_err_t(* motor_position_api_t::positionControl) (motor_position_ctrl_t *const p_ctrl)

Calculate internal position reference

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ ipdSpeedPControl

fsp_err_t(* motor_position_api_t::ipdSpeedPControl) (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)

Calculate iq reference

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | ref_speed_rad | Speed Reference [rad/sec] |
| [in] | speed_rad | Current Speed [rad/sec] |
| [out] | p_iq_ref | Pointer to get iq reference |

### ◆ speedReferencePControlGet

fsp_err_t(* motor_position_api_t::speedReferencePControlGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by P Control

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_speed_ref | Pointer to get speed reference |

### ◆ speedReferenceIpdControlGet

fsp_err_t(* motor_position_api_t::speedReferenceIpdControlGet) (motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)

Get Speed Reference by IPD Control

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_speed_ref | Pointer to get speed reference |

### ◆ speedReferenceFeedforwardGet

fsp_err_t(* motor_position_api_t::speedReferenceFeedforwardGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by Speed Feedforward

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_speed_ref | Pointer to get speed reference |

### ◆ infoGet

fsp_err_t(* motor_position_api_t::infoGet) (motor_position_ctrl_t *const p_ctrl, motor_position_info_t *const p_info)

Get Position information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Pointer to get information |

### ◆ parameterUpdate

fsp_err_t(* motor_position_api_t::parameterUpdate) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Update Parameters for the calculation in the Motor Position Module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_position_instance_t

struct motor_position_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| motor_position_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_position_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_position_api_t const * | p_api | Pointer to the API structure for |

| | | |
|---|---|---|
| | | this instance. |

### Typedef Documentation

#### ◆ motor_position_ctrl_t

| typedef void motor_position_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

### Enumeration Type Documentation

#### ◆ motor_position_ctrl_mode_t

| enum motor_position_ctrl_mode_t |
|---|
| Position Control Mode |

## 5.3.10.10 Motor speed Interface

Interfaces » Motor

### Detailed Description

Interface for motor speed functions.

# Summary

The Motor speed interface for getting the current references from electric current and rotational speed

### Data Structures

| | |
|---|---|
| struct | motor_speed_callback_args_t |
| struct | motor_speed_position_data_t |
| struct | motor_speed_cfg_t |
| struct | motor_speed_api_t |
| struct | motor_speed_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | motor_speed_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | motor_speed_event_t |
| enum | motor_speed_loop_mode_t |
| enum | motor_speed_step_t |

## Data Structure Documentation

### ◆ motor_speed_callback_args_t

| struct motor_speed_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| motor_speed_event_t | event | |

### ◆ motor_speed_position_data_t

| struct motor_speed_position_data_t | | |
|---|---|---|
| Motor speed and position structure | | |
| Data Fields | | |
| motor_speed_step_t | e_step_mode | Select step mode. |
| motor_speed_loop_mode_t | e_loop_mode | Select control mode. |
| int16_t | position_reference_degree | Position reference [degree]. |

### ◆ motor_speed_cfg_t

| struct motor_speed_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| motor_speed_input_t * | st_input |
| | Input data structure for automatic set. |
| | |
| motor_speed_output_t * | st_output |
| | Output data structure for automatic receive. |
| | |
| motor_position_instance_t const * | p_position_instance |
| | Position module instance. |

| | |
|---|---|
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |

### ◆ motor_speed_api_t

| struct motor_speed_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow these APIs. | |
| **Data Fields** | |
| fsp_err_t(* | open )(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(motor_speed_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(motor_speed_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | run )(motor_speed_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | speedReferenceSet )(motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm) |
| | |
| fsp_err_t(* | positionReferenceSet )(motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data) |
| | |
| fsp_err_t(* | parameterSet )(motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input) |
| | |
| fsp_err_t(* | speedControl )(motor_speed_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | parameterGet )(motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output) |
| | |
| fsp_err_t(* | parameterUpdate )(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* motor_speed_api_t::open) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Initialize the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* motor_speed_api_t::close) (motor_speed_ctrl_t *const p_ctrl)

Close (Finish) the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* motor_speed_api_t::reset) (motor_speed_ctrl_t *const p_ctrl)

Reset(Stop) the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ run

fsp_err_t(* motor_speed_api_t::run) (motor_speed_ctrl_t *const p_ctrl)

Activate the motor speed control.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ speedReferenceSet

fsp_err_t(* motor_speed_api_t::speedReferenceSet) (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)

Set (Input) speed reference into the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | speed_reference_rpm | Speed reference [rpm] |

### ◆ positionReferenceSet

fsp_err_t(* motor_speed_api_t::positionReferenceSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)

Set (Input) position reference and control mode

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_position_data | Pointer to structure position data |

### ◆ parameterSet

fsp_err_t(* motor_speed_api_t::parameterSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)

Set (Input) speed parameters into the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_st_input | Pointer to structure to input parameters. |

### ◆ speedControl

fsp_err_t(* motor_speed_api_t::speedControl) (motor_speed_ctrl_t *const p_ctrl)

Calculate current reference

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ parameterGet

| fsp_err_t(* motor_speed_api_t::parameterGet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output) |
|---|

Get speed control output parameters

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_st_output | Pointer to get speed control parameters |

### ◆ parameterUpdate

| fsp_err_t(* motor_speed_api_t::parameterUpdate) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) |
|---|

Update Parameters for the calculation in the motor speed module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure include update parameters. |

### ◆ motor_speed_instance_t

| struct motor_speed_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| motor_speed_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| motor_speed_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| motor_speed_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ motor_speed_ctrl_t

| typedef void motor_speed_ctrl_t |
|---|

Control block. Allocate an instance specific control block to pass into the API calls.

## Enumeration Type Documentation

### ◆ motor_speed_event_t

| enum motor_speed_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| MOTOR_SPEED_EVENT_FORWARD | Event forward speed control. |
| MOTOR_SPEED_EVENT_BACKWARD | Event backward speed control. |
| MOTOR_SPEED_EVENT_ENCODER_CYCLIC | Event encoder cyclic. |
| MOTOR_SPEED_EVENT_ENCODER_ADJUST | Event encoder adjust. |

### ◆ motor_speed_loop_mode_t

| enum motor_speed_loop_mode_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_LOOP_MODE_SPEED | Speed control mode. |
| MOTOR_SPEED_LOOP_MODE_POSITION | Position control mode. |

### ◆ motor_speed_step_t

| enum motor_speed_step_t | |
|---|---|
| Enumerator | |
| MOTOR_SPEED_STEP_DISABLE | Position control works without step mode. |
| MOTOR_SPEED_STEP_ENABLE | Position control works with step mode. |

## 5.3.11 Networking

Interfaces

### Detailed Description

Networking Interfaces.

### Modules

| | |
|---|---|
| | BLE ABS Interface<br>Interface for BLE Abstraction functions. |
| | BLE Interface<br>Interface for Bluetooth Low Energy (BLE) functions. |
| | BLE Mesh Network Interfaces<br>BLE Mesh Network Interfaces. |
| | DA16XXX AT Command Transport Layer<br>Abstraction interface for DA16XXX AT Command functions. |
| | EtherCAT SSC Interface<br>Interface for EtherCAT SSC functions. |
| | Ethernet Interface<br>Interface for Ethernet functions. |
| | Ethernet PHY Interface<br>Interface for Ethernet PHY functions. |
| | Ethernet Switch Interface<br>Interface for Ethernet Switch functions. |
| | GPTP Interface<br>Interface for gPTP timing. |
| | PTP Interface<br>Interface for PTP functions. |
| | WiFi Interface<br>Interface for common WiFi APIs. |

### 5.3.11.1 BLE ABS Interface

Interfaces » Networking

#### Detailed Description

Interface for BLE Abstraction functions.

# Summary

The BLE Abstraction (BLE ABS) interface provides an abstraction layer for various Renesas BLE hardware.

#### Data Structures

| | |
|---|---|
| struct | ble_device_address_t |
| struct | ble_gap_connection_parameter_t |
| struct | ble_gap_connection_phy_parameter_t |
| struct | ble_gap_scan_phy_parameter_t |
| struct | ble_gap_scan_on_t |
| struct | ble_abs_callback_args_t |
| struct | ble_abs_pairing_parameter_t |
| struct | ble_abs_gatt_server_callback_set_t |
| struct | ble_abs_gatt_client_callback_set_t |
| struct | ble_abs_legacy_advertising_parameter_t |
| struct | ble_abs_extend_advertising_parameter_t |
| struct | ble_abs_non_connectable_advertising_parameter_t |
| struct | ble_abs_periodic_advertising_parameter_t |
| struct | ble_abs_scan_phy_parameter_t |
| struct | ble_abs_scan_parameter_t |
| struct | ble_abs_connection_phy_parameter_t |

| | |
|---|---|
| struct | ble_abs_connection_parameter_t |
| struct | ble_abs_cfg_t |
| struct | ble_abs_api_t |
| struct | ble_abs_instance_t |

### Macros

| | |
|---|---|
| #define | BLE_ABS_ADVERTISING_PHY_LEGACY<br>Non-Connectable Legacy Advertising phy setting. |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data) |
| typedef void(* | ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data) |
| typedef void(* | ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data) |
| typedef void(* | ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data) |
| typedef void(* | ble_abs_delete_bond_application_callback_t) (st_ble_dev_addr_t *p_addr) |
| typedef void | ble_abs_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ble_abs_advertising_filter_t |
| enum | ble_abs_local_bond_information_t |
| enum | ble_abs_remote_bond_information_t |
| enum | ble_abs_delete_non_volatile_area_t |

## Data Structure Documentation

### ◆ ble_device_address_t

| struct ble_device_address_t |
|---|
| st_ble_device_address is the type of bluetooth device address(BD_ADDR). |
| Data Fields |

| | | |
|---|---|---|
| uint8_t | addr[BLE_BD_ADDR_LEN] | bluetooth device address. |
| uint8_t | type | the type of bluetooth device address. |

### ◆ ble_gap_connection_parameter_t

| struct ble_gap_connection_parameter_t | | |
|---|---|---|
| ble_gap_connection_parameter_t is Connection parameters included in connection interval, slave latency, supervision timeout, ce length. | | |
| Data Fields | | |
| uint16_t | conn_intv_min | Minimum connection interval. |
| uint16_t | conn_intv_max | Maximum connection interval. |
| uint16_t | conn_latency | Slave latency. |
| uint16_t | sup_to | Supervision timeout. |
| uint16_t | min_ce_length | Minimum CE Length. |
| uint16_t | max_ce_length | Maximum CE Length. |

### ◆ ble_gap_connection_phy_parameter_t

| struct ble_gap_connection_phy_parameter_t | | |
|---|---|---|
| ble_gap_connection_phy_parameter_t is Connection parameters per PHY. | | |
| Data Fields | | |
| uint16_t | scan_intv | Scan interval. |
| uint16_t | scan_window | Scan window. |
| ble_gap_connection_parameter_t * | p_conn_param | Connection interval, slave latency, supervision timeout, and CE length. |

### ◆ ble_gap_scan_phy_parameter_t

| struct ble_gap_scan_phy_parameter_t | | |
|---|---|---|
| Scan parameters per scan PHY. | | |
| Data Fields | | |
| uint8_t | scan_type | Scan type. |
| uint16_t | scan_intv | Scan interval. |
| uint16_t | scan_window | Scan window. |

### ◆ ble_gap_scan_on_t

| struct ble_gap_scan_on_t |
|---|
| Parameters configured when scanning starts. |
| Data Fields |

| | | |
|---|---|---|
| uint8_t | proc_type | Procedure type. |
| uint8_t | filter_dups | Filter duplicates. |
| uint16_t | duration | Scan duration. |
| uint16_t | period | Scan period. |

### ◆ ble_abs_callback_args_t

| struct ble_abs_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware. |
| ble_event_cb_t | ble_abs_event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. Set in ble_abs_api_t::open function in ble_abs_cfg_t. |

### ◆ ble_abs_pairing_parameter_t

| struct ble_abs_pairing_parameter_t | | |
|---|---|---|
| st_ble_abs_pairing_parameter_t includes the pairing parameters. | | |
| Data Fields | | |
| uint8_t | io_capabilitie_local_device | IO capabilities of local device. |
| uint8_t | mitm_protection_policy | MITM protection policy. |
| uint8_t | secure_connection_only | Determine whether to accept only Secure Connections or not. |
| uint8_t | local_key_distribute | Type of keys to be distributed from local device. |
| uint8_t | remote_key_distribute | Type of keys which local device requests a remote device to distribute. |
| uint8_t | maximum_key_size | Maximum LTK size. |
| uint8_t | padding[2] | Reserved. |

### ◆ ble_abs_gatt_server_callback_set_t

| struct ble_abs_gatt_server_callback_set_t | | |
|---|---|---|
| GATT Server callback function and the priority. | | |
| Data Fields | | |
| ble_gatt_server_application_call | gatt_server_callback_function | GATT Server callback function. |

| | | |
|---|---|---|
| back_t | | |
| uint8_t | gatt_server_callback_priority | The priority number of GATT Server callback function. |

### ◆ ble_abs_gatt_client_callback_set_t

| struct ble_abs_gatt_client_callback_set_t | | |
|---|---|---|
| GATT Client callback function and the priority. | | |
| Data Fields | | |
| ble_gatt_client_application_callback_t | gatt_client_callback_function | GATT Client callback function. |
| uint8_t | gatt_client_callback_priority | The priority number of GATT Client callback function. |

### ◆ ble_abs_legacy_advertising_parameter_t

| struct ble_abs_legacy_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_legacy_advertising_parameter_t is the parameters for legacy advertising. | | |
| Data Fields | | |
| ble_device_address_t * | p_peer_address | The remote device address.<br>If the p_peer_address parameter is not NULL, Direct Connectable Advertising is performed to the remote address.<br>If the p_peer_address parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter. |
| uint8_t * | p_advertising_data | Advertising Data.<br>If the p_advertising_data is specified as NULL, Advertising Data is not included in the advertising PDU. |
| uint8_t * | p_scan_response_data | Scan Response Data.<br>If the p_scan_response_data is specified as NULL, Scan Response Data is not included in the advertising PDU. |
| uint32_t | fast_advertising_interval | Advertising with the fast_advertising_interval parameter continues for the period specified by the fast_advertising_period parameter.<br>Time(ms) = |

| | | |
|---|---|---|
| | | fast_advertising_interval * 0.625.<br>If the fast_advertising_period parameter is 0, this parameter is ignored.<br>Valid range is 0x00000020 - 0x00FFFFFF. |
| uint32_t | slow_advertising_interval | After the elapse of the fast_advertising_period, advertising with the slow_advertising_interval parameter continues for the period specified by the slow_advertising_period parameter.<br>Time(ms) = slow_advertising_interval * 0.625.<br>Valid range is 0x00000020 - 0x00FFFFFF. |
| uint16_t | fast_advertising_period | The period which advertising with the fast_advertising_interval parameter continues for.<br>Time = duration * 10ms.<br>After the elapse of the fast_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped.<br>Valid range is 0x0000 - 0xFFFF.<br>If the fast_advertising_period parameter is 0x0000, advertising with the fast_advertising_interval parameter is not performed. |
| uint16_t | slow_advertising_period | The period which advertising with the slow_advertising_interval parameter continues for. Time = duration * 10ms.<br>After the elapse of the slow_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped.<br>Valid range is 0x0000 - 0xFFFF.<br>If the slow_advertising_period parameter is 0x0000, the advertising continues. |
| uint16_t | advertising_data_length | Advertising data length(byte).<br>Valid range is 0-31. |

<table>
<tr><td></td><td></td><td>If the advertising_data_length is 0, Advertising Data is not included in the advertising PDU.</td></tr>
<tr><td>uint16_t</td><td>scan_response_data_length</td><td>Scan response data length (in bytes).<br>Scan Response Data(byte).<br>Valid range is 0-31.<br>If the scan_response_data_length is 0, Scan Response Data is not included in the advertising PDU.</td></tr>
<tr><td>uint8_t</td><td>advertising_channel_map</td><td>The channel map used for the advertising packet transmission.<br>It is a bitwise OR of the following values.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_CH_37(0x01)</td><td>Use 37 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_38(0x02)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_39(0x04)</td><td>Use 39 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_ALL(0x07)</td><td>Use 37 - 39 CH.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>advertising_filter_policy</td><td>Advertising filter policy.<br>If the p_peer_address parameter is NULL, the advertising is performed according to the advertising filter policy.<br>If the p_peer_address parameter is not NULL, this parameter is ignored.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST(0x01)</td><td>Process scan and connection requests from only devices in the White List.</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| uint8_t | own_bluetooth_address_type | Own Bluetooth address type. Select one of the following.<br>macro: description<br>BLE_GAP_ADDR_PUBLIC(0x00): Public Address<br>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02): Resolvable Private Address. If the IRK (Identity Resolving Key) of local device has not been registered in Resolving List, public address is used. |
| uint8_t | own_bluetooth_address[6] | Own Bluetooth address. |
| uint8_t | padding[3] | Reserved. |

◆ **ble_abs_extend_advertising_parameter_t**

| struct ble_abs_extend_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_extend_advertising_parameter_t is the parameters for extended advertising. | | |
| Data Fields | | |
| ble_device_address_t * | p_peer_address | The remote device address. If the p_addr parameter is not NULL, Direct Connectable Advertising is performed to the remote address. If the p_addr parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter. |
| uint8_t * | p_advertising_data | Advertising data. If p_adv_data is specified as NULL, advertising data is not set. |
| uint32_t | fast_advertising_interval | Advertising with the fast_advertising_interval parameter continues for the period specified by the fast_advertising_period parameter. Time(ms) = fast_advertising_interval * |

| | | |
|---|---|---|
| | | 0.625.<br>If the fast_advertising_period parameter is 0, this parameter is ignored.<br>Valid range is 0x00000020 - 0x00FFFFFF. |
| uint32_t | slow_advertising_interval | After the elapse of the fast_advertising_period, advertising with the slow_advertising_interval parameter<br>continues for the period specified by the slow_advertising_period parameter.<br>Time(ms) = slow_advertising_interval * 0.625.<br>Valid range is 0x00000020 - 0x00FFFFFF. |
| uint16_t | fast_advertising_period | The period which advertising with the fast_advertising_interval parameter continues for.<br>Time = duration * 10ms.<br>After the elapse of the fast_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped.<br>Valid range is 0x0000 - 0xFFFF.<br>If the fast_advertising_period parameter is 0x0000, the fast_advertising_interval parameter is ignored. |
| uint16_t | slow_advertising_period | The period which advertising with the slow_advertising_interval parameter continues for.<br>Time = duration * 10ms.<br>After the elapse of the slow_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped.<br>Valid range is 0x0000 - 0xFFFF.<br>If the slow_advertising_period parameter is 0x0000, the advertising continues. |
| uint16_t | advertising_data_length | Advertising data length (in bytes).<br>Valid range is 0-229. |

<table>
<tr><td></td><td></td><td>If the adv_data_length is 0, Advertising Data is not included in the advertising PDU.</td></tr>
<tr><td>uint8_t</td><td>advertising_channel_map</td><td>The channel map used for the advertising packet transmission.<br>It is a bitwise OR of the following values.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_CH_37(0x01)</td><td>Use 37 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_38(0x02)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_39(0x04)</td><td>Use 39 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_ALL(0x07)</td><td>Use 37 - 39 CH.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>advertising_filter_policy</td><td>Advertising filter policy.<br>If the p_peer_address parameter is NULL, the advertising is performed according to the advertising filter policy.<br>If the p_peer_address parameter is not NULL, this parameter is ignored.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST(0x01)</td><td>Process scan and connection requests from only devices in the White List.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address_type</td><td>Own Bluetooth address type.<br>Select one of the following.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)<br>Resolvable Private Address.<br>If the IRK (Identity Resolving Key) of local device has not been registered in Resolving List, public address is used.</td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address[6]</td><td>Own Bluetooth address.</td></tr>
<tr><td>uint8_t</td><td>primary_advertising_phy</td><td>Primary advertising PHY.<br>In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Primary Advertising PHY.<br>When the adv_prop_type field is Legacy Advertising PDU type, this field shall be set to BLE_GAP_ADV_PHY_1M.</td></tr><tr><td>BLE_GAP_ADV_PHY_CD(0x03)</td><td>Use Coded PHY as Primary Advertising PHY.<br>Coding scheme is configured by R_BLE_VS_SetCodingScheme().</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>secondary_advertising_phy</td><td>Secondary advertising Phy.<br>Select one of the following.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x0</td><td>Use 1M PHY as Secondary</td></tr></table></td></tr>
</table>

| | | | |
|---|---|---|---|
| | | 1) | Advertising PHY. |
| | | BLE_GAP_ADV_PHY_2M(0x02) | Use 2M PHY as Secondary Advertising PHY. |
| | | BLE_GAP_ADV_PHY_CD(0x03) | Use Coded PHY(S=8) as Secondary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme(). |
| uint8_t | padding[3] | Reserved. | |

### ◆ ble_abs_non_connectable_advertising_parameter_t

| struct ble_abs_non_connectable_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_non_connectable_advertising_parameter_t is the parameters for non-connectable advertising. | | |
| Data Fields | | |
| ble_device_address_t * | p_peer_address | The remote device address. If the p_peer_address parameter is not NULL, Direct Connectable Advertising is performed to the remote address. If the p_peer_address parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter. |
| uint8_t * | p_advertising_data | Advertising data. If p_adv_data is specified as NULL, advertising data is not set. |
| uint32_t | advertising_interval | Advertising with the advertising_interval parameter continues for the period specified by the duration parameter. Time(ms) = advertising_interval * 0.625. If the duration parameter is 0x0000, the advertising with |

| | | |
|---|---|---|
| | | the advertising_interval parameter continue.<br>Valid range is 0x00000020 - 0x00FFFFFF. |
| uint16_t | advertising_duration | The period which advertising with the advertising_interval parameter continues for.<br>Time = advertising_duration * 10ms.<br>After the elapse of the advertising_duration, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped.<br>Valid range is 0x0000 - 0xFFFF.<br>If the advertising_duration parameter is 0x0000, the advertising continues. |
| uint16_t | advertising_data_length | Advertising data length (in bytes).<br>If the primary_advertising_phy parameter is BLE_ABS_ADVERTISING_PHY_LEGACY(0x00), the valid range is 0-31.<br>If the primary_advertising_phy parameter is the other values, the valid range is 0-1650.<br>If the advertising_data_length parameter is 0, Advertising Data is not included in the advertising PDU. |
| uint8_t | advertising_channel_map | The channel map used for the advertising packet transmission.<br>It is a bitwise OR of the following values.<br>macro: description<br>BLE_GAP_ADV_CH_37(0x01): Use 37 CH.<br>BLE_GAP_ADV_CH_38(0x02): Use 38 CH.<br>BLE_GAP_ADV_CH_39(0x04): Use 39 CH.<br>BLE_GAP_ADV_CH_ALL(0x07): Use 37 - 39 CH. |
| uint8_t | own_bluetooth_address_type | Own Bluetooth address type.<br>Select one of the following. |

| | | |
|---|---|---|
| | | macro / description<br>BLE_GAP_ADDR_PUBLIC(0x00) — Public Address<br>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02) — Resolvable Private Address. If the IRK (Identity Resolving Key) of local device has not been registered in Resolving List, public address is used. |
| uint8_t | own_bluetooth_address[6] | Own Bluetooth address. |
| uint8_t | primary_advertising_phy | Primary advertising PHY.<br>In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.<br>macro / description<br>BLE_ABS_ADVERTISING_PHY_LEGACY(0x00) — Use 1M PHY as Primary Advertising PHY for Non-Connectable Legacy Advertising. If Periodic Advertising is performed, this value shall not set to the adv_phy parameter.<br>BLE_GAP_ADV_PHY_1M(0x01) — Use 1M PHY as Primary Advertising PHY. When the adv_prop_type field is Legacy Advertising PDU type, this field shall be |

| | | |
|---|---|---|
| | | set to BLE_GAP_ADV_PHY_1M.<br>BLE_GAP_ADV_PHY_CD(0x03) Use Coded PHY as Primary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme(). |
| uint8_t | secondary_advertising_phy | Secondary advertising Phy. Select one of the following.<br>macro / description<br>BLE_GAP_ADV_PHY_1M(0x01) Use 1M PHY as Secondary Advertising PHY.<br>BLE_GAP_ADV_PHY_2M(0x02) Use 2M PHY as Secondary Advertising PHY.<br>BLE_GAP_ADV_PHY_CD(0x03) Use Coded PHY(S=8) as Secondary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme(). |
| uint8_t | padding[2] | Reserved. |

### ◆ ble_abs_periodic_advertising_parameter_t

| struct ble_abs_periodic_advertising_parameter_t | | |
|---|---|---|
| st_ble_abs_periodic_advertising_parameter_t is the parameters for periodic advertising. | | |
| Data Fields | | |
| ble_abs_non_connectable_advertising_parameter_t | advertising_parameter | Advertising parameters. |
| uint8_t * | p_periodic_advertising_data | Periodic advertising data. If p_perd_adv_data is specified as NULL, periodic advertising data |

| | | |
|---|---|---|
| | | is not set. |
| uint16_t | periodic_advertising_interval | Periodic advertising interval.<br>Time(ms) = periodic_advertising_interval * 1.25.<br>Valid range is 0x0006 - 0xFFFF. |
| uint16_t | periodic_advertising_data_length | Periodic advertising data length (in bytes).<br>Valid range is 0 - 1650.<br>If the periodic_advertising_data_length is 0, Periodic Advertising Data is not included in the advertising PDU. |

### ◆ ble_abs_scan_phy_parameter_t

| struct ble_abs_scan_phy_parameter_t | | |
|---|---|---|
| st_ble_abs_scan_phy_parameter_t is the phy parameters for scan. | | |
| Data Fields | | |
| uint16_t | fast_scan_interval | Fast scan interval.<br>Interval(ms) = fast_scan_interval * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| uint16_t | slow_scan_interval | Slow Scan interval.<br>Slow Scan interval(ms) = slow_scan_interval * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| uint16_t | fast_scan_window | Fast Scan window.<br>Fast Scan window(ms) = fast_scan_window * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| uint16_t | slow_scan_window | Slow Scan window.<br>Slow Scan window(ms) = slow_scan_window * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| uint8_t | scan_type | Scan type.<br>macro: description<br>BLE_GAP_SCAN_PASSIVE(0x00): Passive Scan.<br>BLE_GAP_SCAN_ACTIVE(0x01): Active Scan. |
| uint8_t | padding[3] | Reserved. |

### ◆ ble_abs_scan_parameter_t

| struct ble_abs_scan_parameter_t | | |
|---|---|---|
| st_ble_abs_scan_parameter_t is the parameters for scan. | | |
| Data Fields | | |
| ble_abs_scan_phy_parameter_t * | p_phy_parameter_1M | Scan parameters for receiving the advertising packets in 1M PHY.<br>In case of not receiving the advertising packets in 1M PHY, this field is specified as NULL.<br>p_phy_parameter_1M or p_phy_parameter_coded field shall be set to scan parameters. |
| ble_abs_scan_phy_parameter_t * | p_phy_parameter_coded | Scan parameters for receiving the advertising packets in Coded PHY.<br>In case of not receiving the advertising packets in Coded PHY, this field is specified as NULL.<br>p_phy_parameter_1M or p_phy_parameter_coded field shall be set to scan parameters. |
| uint8_t * | p_filter_data | Data for Advertising Data filtering.<br>The p_filter_data parameter is used for the advertising data in single advertising report.<br>The advertising data composed of multiple advertising reports is not filtered by this parameter.<br>If the p_filter_data parameter is specified as NULL, the filtering is not done. |
| uint16_t | fast_scan_period | The period which scan with the fast scan interval/fast scan window continues for.<br>Time(ms) = fast_scan_period * 10.<br>Valid range is 0x0000 - 0xFFFF.<br>If the fast_scan_period parameter is 0x0000, scan with the fast scan interval/fast scan window is not performed.<br>After the elapse of the fast_scan_period, BLE_GAP_EVENT_SCAN_TO event notifies that the scan has stopped. |
| uint16_t | slow_scan_period | The period which scan with the slow scan interval/slow scan |

<table>
<tr><td></td><td></td><td>window continues for.<br>Time = slow_scan_period * 10ms.<br>Valid range is 0x0000 - 0xFFFF.<br>If the slow_scan_period parameter is 0x0000, the scan continues.<br>After the elapse of the slow_scan_period, BLE_GAP_EVENT_SCAN_TO event notifies that the scan has stopped.</td></tr>
<tr><td>uint16_t</td><td>filter_data_length</td><td>The length of the data specified by the p_filter_data parameter.<br>Valid range is 0x0000-0x0010.<br>If the filter_data_length parameter is 0, the filtering is not done.</td></tr>
<tr><td>uint8_t</td><td>device_scan_filter_policy</td><td>Scan Filter Policy. Select one of the following.<br>Address type setting (Field [7:4])<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Use Random Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>If the IRK (Identity Resolving Key) of local device has been registered in Resolving list, use RPA. If not, use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)</td><td>If the IRK (Identity Resolving Key) of local device has been registered in Resolving list, use RPA. If not, use</td></tr></table></td></tr>
</table>

Random Address.

White list setting (Field [3:0])

| macro | description |
| --- | --- |
| BLE_GAP_SCAN_ALLOW_ADV_ALL(0x00) | Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device. |
| BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01) | Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not addressed to local device is ignored. |
| BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED(0x02) | Accept all advertising and scan response PDUs except directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target |

| | | | |
|---|---|---|---|
| | | | address is the local resolvable private address are accepted. |
| | | BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST(0x03) | Accept all advertising and scan response PDUs. The following are excluded:<br><br>Advertising and scan response PDUs where the advertiser's identity address is not in the White List.<br><br>Directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted. |
| uint8_t | filter_duplicate | Filter duplicates.<br>Maximum number of filtered devices is 8.<br>The 9th and subsequent devices are not filtered by this parameter. | |

| | | |
|---|---|---|
| | | macro / description<br>BLE_GAP_SCAN_FILT_DUPLIC_DISABLE(0x00) — Duplicate filter disabled.<br>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE(0x01) — Duplicate filter enabled.<br>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02)) — Duplicate filtering enabled, reset for each scan period. |
| uint8_t | filter_ad_type | The AD type of the data specified by the p_filter_data parameter.<br>The AD type identifier values are defined in Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers). |
| uint8_t | padding[3] | Padding. |

### ◆ ble_abs_connection_phy_parameter_t

| struct ble_abs_connection_phy_parameter_t | | |
|---|---|---|
| st_ble_abs_connection_phy_parameter_t is the phy parameters for create connection. | | |
| Data Fields | | |
| uint16_t | connection_interval | Connection interval.<br>Time(ms) = connection_interval * 1.25.<br>Valid range is 0x0006 - 0x0C80. |
| uint16_t | connection_slave_latency | Slave latency.<br>Valid range is 0x0000 - 0x01F3. |
| uint16_t | supervision_timeout | Supervision timeout.<br>Time(ms) = supervision_timeout * 10.<br>Valid range is 0x000A - 0x0C80. |
| uint8_t | padding[2] | Padding. |

### ◆ ble_abs_connection_parameter_t

| struct ble_abs_connection_parameter_t |
|---|
| st_ble_abs_connection_parameter_t is the parameters for create connection. |
| Data Fields |

<table>
<tr><td>ble_abs_connection_phy_parameter_t *</td><td>p_connection_phy_parameter_1M</td><td>Connection interval, slave latency, supervision timeout for 1M PHY.<br>The p_connection_phy_parameter_1M is specified as NULL, a connection request is not sent with 1M PHY.</td></tr>
<tr><td>ble_abs_connection_phy_parameter_t *</td><td>p_connection_phy_parameter_2M</td><td>Connection interval, slave latency, supervision timeout for 2M PHY.<br>The p_connection_phy_parameter_2M is specified as NULL, a connection request is not sent with 2M PHY.</td></tr>
<tr><td>ble_abs_connection_phy_parameter_t *</td><td>p_connection_phy_parameter_coded</td><td>Connection interval, slave latency, supervision timeout for Coded PHY.<br>The p_connection_phy_parameter_coded is specified as NULL, a connection request is not sent with Coded PHY.</td></tr>
<tr><td>ble_device_address_t *</td><td>p_device_address</td><td>Address of the device to be connected.<br>If the filter field is BLE_GAP_INIT_FILT_USE_WLST(0x01), this parameter is ignored and please fill p_device_address.addr with 0x00.</td></tr>
<tr><td>uint8_t</td><td>filter_parameter</td><td>The filter field specifies whether the White List is used or not, when connecting with a remote device.<br>Address type setting (Field [7:4])<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Use Random Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>If the IRK (Identity Resolving Key) of local device has been registered in Resolving list,</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td>use RPA. If not, use Public Address.<br><br>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) If the IRK (Identity Resolving Key) of local device has been registered in Resolving list, use RPA. If not, use Random Address.<br><br>White list setting (Field [3:0])<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_INIT_FILT_USE_ADDR(0x00)</td><td>White List is not used.<br>The remote device to be connected is specified by the p_addr field is used.</td></tr><tr><td>BLE_GAP_INIT_FILT_USE_WLST(0x01)</td><td>White List is used.<br>The remote device registered in White List is connected with local device.<br>The p_addr field is ignored.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>connection_timeout</td><td>The time(sec) to cancel the create connection request.<br>Valid range is 0 <= connection_timeout <= 10.<br>If the connection_timeout field is 0, the create connection request is not canceled.</td></tr>
<tr><td>uint8_t</td><td>padding[2]</td><td>Padding.</td></tr>
</table>

### ◆ ble_abs_cfg_t

| struct ble_abs_cfg_t |
|---|

BLE ABS configuration parameters.

| **Data Fields** | |
|---|---|
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| ble_gap_application_callback_t | gap_callback |
| | GAP callback function. |
| | |
| ble_vendor_specific_application_callback_t | vendor_specific_callback |
| | Vendor Specific callback function. |
| | |
| ble_abs_gatt_server_callback_set_t * | p_gatt_server_callback_list |
| | GATT Server callback set. |
| | |
| uint8_t | gatt_server_callback_list_number |
| | The number of GATT Server callback functions. |
| | |
| ble_abs_gatt_client_callback_set_t * | p_gatt_client_callback_list |
| | GATT Client callback set. |
| | |
| uint8_t | gatt_client_callback_list_number |
| | The number of GATT Client callback functions. |
| | |
| ble_abs_pairing_parameter_t * | p_pairing_parameter |
| | Pairing parameters. |

| | |
|---|---|
| | |
| flash_instance_t const * | p_flash_instance |
| | Pointer to flash instance. |
| | |
| timer_instance_t const * | p_timer_instance |
| | Pointer to timer instance. |
| | |
| void(* | p_callback )(ble_abs_callback_args_t *p_args) |
| | Callback provided when a BLE ISR occurs. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ channel

| uint32_t ble_abs_cfg_t::channel |
|---|
| Select a channel corresponding to the channel number of the hardware.<br>the parameters for initialization. |

### ◆ ble_abs_api_t

| struct ble_abs_api_t | |
|---|---|
| BLE ABS functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ble_abs_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | reset )(ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback) |
| | |
| fsp_err_t(* | startLegacyAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startExtendedAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startNonConnectableAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startPeriodicAdvertising )(ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter) |
| | |
| fsp_err_t(* | startScanning )(ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter) |
| | |
| fsp_err_t(* | createConnection )(ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter) |
| | |
| fsp_err_t(* | setLocalPrivacy )(ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode) |
| | |
| fsp_err_t(* | startAuthentication )(ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle) |
| | |
| fsp_err_t(* | deleteBondInformation )(ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter) |
| | |
| fsp_err_t(* | importKeyInformation )(ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk) |
| | |
| fsp_err_t(* | exportKeyInformation )(ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, |

| | |
|---|---|
| | uint8_t *p_local_csrk) |

## Field Documentation

### ◆ open

fsp_err_t(* ble_abs_api_t::open) (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg)

Initialize the BLE ABS in register start mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ble_abs_api_t::close) (ble_abs_ctrl_t *const p_ctrl)

Close the BLE ABS.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reset

fsp_err_t(* ble_abs_api_t::reset) (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback)

Close the BLE ABS.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | init_callback | callback function to initialize Host Stack. |

### ◆ startLegacyAdvertising

fsp_err_t(* ble_abs_api_t::startLegacyAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter)

Start Legacy Connectable Advertising.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for Legacy Advertising. |

### ◆ startExtendedAdvertising

fsp_err_t(* ble_abs_api_t::startExtendedAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter)

Start Extended Connectable Advertising.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for extend Advertising. |

### ◆ startNonConnectableAdvertising

fsp_err_t(* ble_abs_api_t::startNonConnectableAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter)

Start Non-Connectable Advertising.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for non-connectable Advertising. |

### ◆ startPeriodicAdvertising

fsp_err_t(* ble_abs_api_t::startPeriodicAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter)

Start Periodic Advertising.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_advertising_parameter | Pointer to Advertising parameters for periodic Advertising. |

### ◆ startScanning

fsp_err_t(* ble_abs_api_t::startScanning) (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter)

Start scanning.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_scan_parameter | Pointer to scan parameter. |

### ◆ createConnection

fsp_err_t(* ble_abs_api_t::createConnection) (ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter)

Request create connection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_connection_parameter | Pointer to connection parameter. |

### ◆ setLocalPrivacy

fsp_err_t(* ble_abs_api_t::setLocalPrivacy) (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode)

Configure local device privacy.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_lc_irk | Pointer to IRK (Identity Resolving Key) to be registered in the resolving list. |
| [in] | privacy_mode | privacy_mode privacy mode. |

### ◆ startAuthentication

fsp_err_t(* ble_abs_api_t::startAuthentication) (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle)

Start pairing or encryption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | connection_handle | Connection handle identifying the remote device. |

### ◆ deleteBondInformation

fsp_err_t(* ble_abs_api_t::deleteBondInformation) (ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter)

Delete bond information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_bond_information_parameter | Pointer to bond information parameter. |

### ◆ importKeyInformation

fsp_err_t(* ble_abs_api_t::importKeyInformation) (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Import local identity address, keys information to local storage.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_local_identity_address | Pointer to local identiry address. |
| [in] | uint8_t | p_local_irk Pointer to local IRK (Identity Resolving Key) |
| [in] | uint8_t | p_local_csrk Pointer to local CSRK |

### ◆ exportKeyInformation

fsp_err_t(* ble_abs_api_t::exportKeyInformation) (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Export local identity address, keys information from local storage.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_local_identity_address | Pointer to local identiry address. |
| [out] | uint8_t | p_local_irk Pointer to local IRK (Identity Resolving Key) |
| [out] | uint8_t | p_local_csrk Pointer to local CSRK |

### ◆ ble_abs_instance_t

struct ble_abs_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| ble_abs_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ble_abs_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ble_abs_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ble_gap_application_callback_t

| typedef void(* ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data) |
|---|
| ble_gap_application_callback_t is the GAP Event callback function type. |

### ◆ ble_vendor_specific_application_callback_t

| typedef void(* ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data) |
|---|
| ble_vendor_specific_application_callback_t is the Vendor Specific Event callback function type. |

### ◆ ble_gatt_server_application_callback_t

| typedef void(* ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data) |
|---|
| ble_gatt_server_application_callback_t is the GATT Server Event callback function type. |

### ◆ ble_gatt_client_application_callback_t

| typedef void(* ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data) |
|---|
| ble_gatt_client_application_callback_t is the GATT Server Event callback function type. |

### ◆ ble_abs_delete_bond_application_callback_t

| typedef void(* ble_abs_delete_bond_application_callback_t) (st_ble_dev_addr_t *p_addr) |
|---|
| ble_abs_delete_bond_application_callback_t is the delete bond information Event callback function type. |

### ◆ ble_abs_ctrl_t

| typedef void ble_abs_ctrl_t |
|---|
| BLE ABS control block. Allocate an instance specific control block to pass into the BLE ABS API calls. |

## Enumeration Type Documentation

### ◆ ble_abs_advertising_filter_t

| enum ble_abs_advertising_filter_t | |
|---|---|
| Advertising Filter Policy | |
| Enumerator | |
| BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY | Receive a connect request from all devices. |
| BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST | Receive a connect request from only the devices registered in White List. |

### ◆ ble_abs_local_bond_information_t

| enum ble_abs_local_bond_information_t | |
|---|---|
| Local keys delete policy | |
| Enumerator | |
| BLE_ABS_LOCAL_BOND_INFORMATION_NONE | Delete no local keys. |
| BLE_ABS_LOCAL_BOND_INFORMATION_ALL | Delete all local keys. |

### ◆ ble_abs_remote_bond_information_t

| enum ble_abs_remote_bond_information_t | |
|---|---|
| Remote keys delete policy | |
| Enumerator | |
| BLE_ABS_REMOTE_BOND_INFORMATION_NONE | Delete no remote device keys. |
| BLE_ABS_REMOTE_BOND_INFORMATION_SPECIFIED | Delete the keys specified by the device address. |
| BLE_ABS_REMOTE_BOND_INFORMATION_ALL | Delete all remote device keys. |

### ◆ ble_abs_delete_non_volatile_area_t

| enum ble_abs_delete_non_volatile_area_t | |
|---|---|
| Deletion policy for non-volatile memory | |
| Enumerator | |
| BLE_ABS_DELETE_NON_VOLATILE_AREA_DISABLE | Delete no keys stored in storage. |
| BLE_ABS_DELETE_NON_VOLATILE_AREA_ENABLE | Delete the keys stored in storage. |

## 5.3.11.2 BLE Interface

Interfaces » Networking

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_Open (void)<br>Open the BLE protocol stack. More... |
| ble_status_t | R_BLE_Close (void)<br>Close the BLE protocol stack. More... |
| ble_status_t | R_BLE_Execute (void)<br>Execute the BLE task. More... |
| uint32_t | R_BLE_IsTaskFree (void)<br>Check the BLE task queue is free or not. More... |
| ble_status_t | R_BLE_SetEvent (ble_event_cb_t cb)<br>Set event. More... |
| uint32_t | R_BLE_GetVersion (void)<br>Get the BLE FIT module version. More... |
| uint32_t | R_BLE_GetLibType (void)<br>Get the type of BLE protocol stack library. More... |

### Detailed Description

Interface for Bluetooth Low Energy (BLE) functions.

# Summary

The BLE interface for the Bluetooth Low Energy (BLE) peripheral provides BLE functionality.

### Modules

| | |
|---|---|
| | GAP |
| | ISO |
| | GATT_COMMON |
| | GATT_SERVER |
| | GATT_CLIENT |
| | L2CAP |
| | VS |

### Macros

| | |
|---|---|
| #define | BLE_VERSION_MAJOR |
| #define | BLE_VERSION_MINOR |
| #define | BLE_LIB_EXTENDED |
| #define | BLE_LIB_BALANCE |
| #define | BLE_LIB_COMPACT |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_event_cb_t) (void)<br>ble_event_cb_t is the callback function type for R_BLE_SetEvent(). More... |

### Macro Definition Documentation

### ◆ BLE_VERSION_MAJOR

| #define BLE_VERSION_MAJOR |
| --- |
| BLE Module Major Version. |

### ◆ BLE_VERSION_MINOR

| #define BLE_VERSION_MINOR |
| --- |
| BLE Module Minor Version. |

### ◆ BLE_LIB_EXTENDED

| #define BLE_LIB_EXTENDED |
| --- |
| BLE Protocol Stack Library Extended type. |

### ◆ BLE_LIB_BALANCE

| #define BLE_LIB_BALANCE |
| --- |
| BLE Protocol Stack Library Balance type. |

### ◆ BLE_LIB_COMPACT

| #define BLE_LIB_COMPACT |
| --- |
| BLE Protocol Stack Library Compacy type. |

## Typedef Documentation

### ◆ ble_event_cb_t

| ble_event_cb_t |
| --- |
| ble_event_cb_t is the callback function type for R_BLE_SetEvent().<br><br>**Returns**<br>none |

## Function Documentation

### ◆ R_BLE_Open()

| ble_status_t R_BLE_Open ( void ) |
|---|
| Open the BLE protocol stack.<br>This function should be called once before using the BLE protocol stack.<br><br>**Return values**<br>BLE_SUCCESS(0x0000) : Success |

### ◆ R_BLE_Close()

| ble_status_t R_BLE_Close ( void ) |
|---|
| Close the BLE protocol stack.<br>This function should be called once to close the BLE protocol stack.<br><br>**Return values**<br>BLE_SUCCESS(0x0000) : Success |

### ◆ R_BLE_Execute()

| ble_status_t R_BLE_Execute ( void ) |
|---|
| Execute the BLE task.<br>This handles all the task queued in the BLE protocol stack internal task queue and return. This function should be called repeatedly in the main loop.<br><br>**Return values**<br>BLE_SUCCESS(0x0000) : Success |

### ◆ R_BLE_IsTaskFree()

uint32_t R_BLE_IsTaskFree ( void )

Check the BLE task queue is free or not.

This function returns the BLE task queue free status. When this function returns 0x0, call R_BLE_Execute() to execute the BLE task.

**Return values**

| | |
|---|---|
| 0x0 | BLE task queue is not free |
| 0x1 | BLE task queue is free |

### ◆ R_BLE_SetEvent()

ble_status_t R_BLE_SetEvent ( ble_event_cb_t *cb*)

Set event.

This function add an event in the BLE protocol stack internal queue. The event is handled in R_BLE_Execute just like Bluetooth event. This function is intended to be called in hardware interrupt context. Even if calling this function with the same cb before the cb is invoked, only one event is registered. The maximum number of the events can be registered at a time is eight.

**Parameters**

| | |
|---|---|
| cb | The callback for the event. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_ALREADY_IN_PROGRESS(0x000A) | The event already registered with the callback. |
| BLE_ERR_CONTEXT_FULL(0x000B) | No free slot for the event. |

### ◆ R_BLE_GetVersion()

uint32_t R_BLE_GetVersion ( void )

Get the BLE FIT module version.

This function returns the BLE FIT module version.
The major version(BLE_VERSION_MAJOR) is contained in the two most significant bytes, and the minor version(BLE_VERSION_MINOR) occupies the remaining two bytes.

**Return values**

| | |
|---|---|
| BLE_VERSION_MAJOR \| BLE_VERSION_MINOR | |

### ◆ R_BLE_GetLibType()

| uint32_t R_BLE_GetLibType ( void ) |
|---|
| Get the type of BLE protocol stack library.<br>This function returns the type of BLE protocol stack library.<br><br>**Return values**<br>BLE_LIB_EXTENDED(0x00) — Extended<br>BLE_LIB_BALANCE(0x01) — Balance<br>BLE_LIB_COMPACT(0x02) — Compact |

## GAP

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_GAP_Init (ble_gap_app_cb_t gap_cb)<br>Initialize the Host Stack. More... |
| ble_status_t | R_BLE_GAP_Terminate (void)<br>Terminate the Host Stack. More... |
| ble_status_t | R_BLE_GAP_UpdConn (uint16_t conn_hdl, uint8_t mode, uint16_t accept, st_ble_gap_conn_param_t *p_conn_updt_param)<br>Update the connection parameters. More... |
| ble_status_t | R_BLE_GAP_SetDataLen (uint16_t conn_hdl, uint16_t tx_octets, uint16_t tx_time)<br>Update the packet size and the packet transmit time. More... |
| ble_status_t | R_BLE_GAP_Disconnect (uint16_t conn_hdl, uint8_t reason)<br>Disconnect the link. More... |
| ble_status_t | R_BLE_GAP_SetPhy (uint16_t conn_hdl, st_ble_gap_set_phy_param_t *p_phy_param)<br>Set the phy for connection. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_SetDefPhy (st_ble_gap_set_def_phy_param_t *p_def_phy_param)<br>Set the default phy which allows remote device to change. More... |
| ble_status_t | R_BLE_GAP_SetPrivMode (st_ble_dev_addr_t *p_addr, uint8_t *p_privacy_mode, uint8_t device_num)<br>Set the privacy mode. More... |
| ble_status_t | R_BLE_GAP_ConfWhiteList (uint8_t op_code, st_ble_dev_addr_t *p_addr, uint8_t device_num)<br>Set White List. More... |
| ble_status_t | R_BLE_GAP_GetVerInfo (void)<br>Get the version number of the Controller and the host stack. More... |
| ble_status_t | R_BLE_GAP_ReadPhy (uint16_t conn_hdl)<br>Get the phy settings. More... |
| ble_status_t | R_BLE_GAP_ConfRslvList (uint8_t op_code, st_ble_dev_addr_t *p_addr, st_ble_gap_rslv_list_key_set_t *p_peer_irk, uint8_t device_num)<br>Set Resolving List. More... |
| ble_status_t | R_BLE_GAP_EnableRpa (uint8_t enable)<br>Enable/Disable address resolution and generation of a resolvable private address. More... |
| ble_status_t | R_BLE_GAP_SetRpaTo (uint16_t rpa_timeout)<br>Set the update time of resolvable private address. More... |
| ble_status_t | R_BLE_GAP_ReadRpa (st_ble_dev_addr_t *p_addr)<br>Get the resolvable private address of local device. More... |
| ble_status_t | R_BLE_GAP_ReadRssi (uint16_t conn_hdl)<br>Get RSSI. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_ReadChMap (uint16_t conn_hdl)<br>Get the Channel Map. More... |
| ble_status_t | R_BLE_GAP_SetRandAddr (uint8_t *p_random_addr)<br>Set a random address. More... |
| ble_status_t | R_BLE_GAP_SetAdvParam (st_ble_gap_adv_param_t *p_adv_param)<br>Set advertising parameters. More... |
| ble_status_t | R_BLE_GAP_SetAdvSresData (st_ble_gap_adv_data_t *p_adv_srsp_data)<br>Set advertising data/scan response data/periodic advertising data. More... |
| ble_status_t | R_BLE_GAP_StartAdv (uint8_t adv_hdl, uint16_t duration, uint8_t max_extd_adv_evts)<br>Start advertising. More... |
| ble_status_t | R_BLE_GAP_StopAdv (uint8_t adv_hdl)<br>Stop advertising. More... |
| ble_status_t | R_BLE_GAP_SetPerdAdvParam (st_ble_gap_perd_adv_param_t *p_perd_adv_param)<br>Set periodic advertising parameters. More... |
| ble_status_t | R_BLE_GAP_StartPerdAdv (uint8_t adv_hdl)<br>Start periodic advertising. More... |
| ble_status_t | R_BLE_GAP_StopPerdAdv (uint8_t adv_hdl)<br>Stop periodic advertising. More... |
| ble_status_t | R_BLE_GAP_GetRemainAdvBufSize (uint16_t *p_remain_adv_data_size, uint16_t *p_remain_perd_adv_data_size)<br>Get buffer size for advertising data/scan response data/periodic advertising data in the Controller. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_RemoveAdvSet (uint8_t op_code, uint8_t adv_hdl)<br>Delete advertising set. More... |
| ble_status_t | R_BLE_GAP_CreateConn (st_ble_gap_create_conn_param_t *p_param)<br>Request for a link establishment. More... |
| ble_status_t | R_BLE_GAP_CancelCreateConn (void)<br>Cancel the request for a link establishment. More... |
| ble_status_t | R_BLE_GAP_SetChMap (uint8_t *p_channel_map)<br>Set the Channel Map. More... |
| ble_status_t | R_BLE_GAP_StartScan (st_ble_gap_scan_param_t *p_scan_param, st_ble_gap_scan_on_t *p_scan_enable)<br>Set scan parameter and start scan. More... |
| ble_status_t | R_BLE_GAP_StopScan (void)<br>Stop scan. More... |
| ble_status_t | R_BLE_GAP_CreateSync (st_ble_dev_addr_t *p_addr, uint8_t adv_sid, uint16_t skip, uint16_t sync_to)<br>Request for a periodic sync establishment. More... |
| ble_status_t | R_BLE_GAP_CancelCreateSync (void)<br>Cancel the request for a periodic sync establishment. More... |
| ble_status_t | R_BLE_GAP_TerminateSync (uint16_t sync_hdl)<br>Terminate the periodic sync. More... |
| ble_status_t | R_BLE_GAP_ConfPerdAdvList (uint8_t op_code, st_ble_dev_addr_t *p_addr, uint8_t *p_adv_sid_set, uint8_t device_num)<br>Set Periodic Advertiser List. More... |
| ble_status_t | R_BLE_GAP_AuthorizeDev (uint16_t conn_hdl, uint8_t author_flag) |

| | |
|---|---|
| | Authorize a remote device. More... |
| ble_status_t | R_BLE_GAP_GetRemDevInfo (uint16_t conn_hdl)<br>Get the information about remote device. More... |
| ble_status_t | R_BLE_GAP_SetPairingParams (st_ble_gap_pairing_param_t *p_pair_param)<br>Set the parameters using pairing. More... |
| ble_status_t | R_BLE_GAP_SetLocIdInfo (st_ble_dev_addr_t *p_lc_id_addr, uint8_t *p_lc_irk)<br>Set the IRK and the identity address distributed to a remote device. More... |
| ble_status_t | R_BLE_GAP_SetLocCsrk (uint8_t *p_local_csrk)<br>Set the CSRK distributed to a remote device. More... |
| ble_status_t | R_BLE_GAP_StartPairing (uint16_t conn_hdl)<br>Start pairing. More... |
| ble_status_t | R_BLE_GAP_ReplyPairing (uint16_t conn_hdl, uint8_t response)<br>Reply the pairing request from a remote device. More... |
| ble_status_t | R_BLE_GAP_StartEnc (uint16_t conn_hdl)<br>Encryption the link. More... |
| ble_status_t | R_BLE_GAP_ReplyPasskeyEntry (uint16_t conn_hdl, uint32_t passkey, uint8_t response)<br>Reply the passkey entry request. More... |
| ble_status_t | R_BLE_GAP_ReplyNumComp (uint16_t conn_hdl, uint8_t response)<br>Reply the numeric comparison request. More... |
| ble_status_t | R_BLE_GAP_NotifyKeyPress (uint16_t conn_hdl, uint8_t key_press)<br>Notify the input key type which a remote device inputs in the passkey entry. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_GetDevSecInfo (uint16_t conn_hdl, st_ble_gap_auth_info_t *p_sec_info)<br>Get the security information about the remote device. More... |
| ble_status_t | R_BLE_GAP_ReplyExKeyInfoReq (uint16_t conn_hdl)<br>Distribute the keys of local device. More... |
| ble_status_t | R_BLE_GAP_SetRemOobData (st_ble_dev_addr_t *p_addr, uint8_t oob_data_flag, st_ble_gap_oob_data_t *p_oob)<br>Set the oob data from a remote device. More... |
| ble_status_t | R_BLE_GAP_CreateScOobData (void)<br>Create data for oob in secure connection. More... |
| ble_status_t | R_BLE_GAP_SetBondInfo (st_ble_gap_bond_info_t *p_bond_info, uint8_t device_num, uint8_t *p_set_num)<br>Set the bonding information stored in non-volatile memory to the host stack. More... |
| void | R_BLE_GAP_DeleteBondInfo (int32_t local, int32_t remote, st_ble_dev_addr_t *p_addr, ble_gap_del_bond_cb_t gap_del_bond_cb)<br>This function deletes the bonding information in Host Stack.<br>When a function for deleting the bonding information stored in non-volatile area is registered by the gap_del_bond_cb parameter, it is deleted as well as the bonding information in Host Stack. More... |
| ble_status_t | R_BLE_GAP_ReplyLtkReq (uint16_t conn_hdl, uint16_t ediv, uint8_t *p_peer_rand, uint8_t response)<br>Reply the LTK request from a remote device. More... |
| ble_status_t | R_BLE_GAP_SetCteConnlessParam (st_ble_gap_cte_connless_t *p_cte_param)<br>Set the parameters for the transmission of Constant Tone Extensions in any periodic advertising. More... |
| ble_status_t | R_BLE_GAP_EnableCteConnless (uint16_t adv_hdl, uint8_t enable)<br>Enable or disable Constant Tone Extensions in periodic advertising identified by the adv_hdl. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_StartCteConnlessRecv (st_ble_gap_cte_connless_recv_t *p_cte_recv)<br>Enable sampling received Constant Tone Extension fields. More... |
| ble_status_t | R_BLE_GAP_StopCteConnlessRecv (uint16_t sync_hdl)<br>Disable sampling received Constant Tone Extension fields. More... |
| ble_status_t | R_BLE_GAP_SetCteConnParam (st_ble_gap_cte_conn_t *p_cte_param)<br>Set the parameters for the transmission of Constant Tone Extensions in ACL link. More... |
| ble_status_t | R_BLE_GAP_EnableCteConnRsp (uint16_t conn_hdl, uint8_t enable)<br>Enable or disable Constant Tone Extensions Transmission in ACL link by conn_hdl. More... |
| ble_status_t | R_BLE_GAP_SetCteConnRecvParam (st_ble_gap_cte_conn_rx_param_t *p_cte_param)<br>Set the parameters for the receiving of Constant Tone Extensions in ACL link. and start sampling. More... |
| ble_status_t | R_BLE_GAP_StopCteConnRecvSampling (uint16_t conn_hdl)<br>Stop sampling of Constant Tone Extensions on the specified connection. More... |
| ble_status_t | R_BLE_GAP_StartCteConnReq (st_ble_gap_cte_conn_req_t *p_req)<br>Set the parameters and start sending request of Constant Tone Extensions in ACL link to peer. More... |
| ble_status_t | R_BLE_GAP_StopCteConnReq (uint16_t handle)<br>Stop sending request of Constant Tone Extensions in ACL link to peer. More... |
| ble_status_t | R_BLE_GAP_SetDefaultSubrate (st_ble_gap_subrate_param_t *p_subrate_param)<br>Set the initial values for the acceptable parameters for subrating requests,. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_RequestSubrate (uint16_t conn_hdl, st_ble_gap_subrate_param_t *p_subrate_param)<br>Request a change to the subrating factor other parameters applied to an existing connection using the Connection Subrate Update procedure. More... |
| ble_status_t | R_BLE_GAP_StartPerdAdvSetInfoTransfer (uint16_t adv_hdl, uint16_t conn_hdl, uint16_t service_data)<br>This function starts Periodic advertising adv set info transfer to the connection. More... |
| ble_status_t | R_BLE_GAP_StartPerdAdvSyncTransfer (uint16_t sync_hdl, uint16_t conn_hdl, uint16_t service_data)<br>This function starts Periodic advertising sync transfer. More... |
| ble_status_t | R_BLE_GAP_SetPerdAdvSyncTransferParam (uint16_t conn_hdl, st_ble_gap_past_param_t *p_past_param)<br>This function starts to accept Periodic advertising sync transfer from the connection. More... |
| ble_status_t | R_BLE_GAP_SetDefPerdAdvSyncTransferParam (st_ble_gap_past_param_t *p_past_param)<br>This function set the default parameter of Periodic advertising sync transfer for all subsequent connection. It does not affect any existing connection. More... |
| ble_status_t | R_BLE_GAP_ReadAntennaInfo (void)<br>This function read the switching rates, the sampling rates, the number of antennae, and the maximum length of a transmitted Constant Tone Extension. More... |
| ble_status_t | R_BLE_GAP_ReceiverTest (st_ble_gap_recv_test_param_t *p_rx_test_param)<br>Start a test where the DUT receives test reference packets at a fixed interval. The tester generates the test reference packets. More... |
| ble_status_t | R_BLE_GAP_TransmitterTest (st_ble_gap_trans_test_param_t *p_tx_test_param)<br>Start a test where the DUT generates test reference packets at a fixed interval. The Controller shall transmit at the power level indicated by the TX_Power_Level parameter. More... |

| | |
|---|---|
| ble_status_t | R_BLE_GAP_ModifySleepClockAccuracy (uint8_t act)<br>request that the Controller changes its sleep clock accuracy for testing purposes. It should not be used under other circumstances. More... |
| ble_status_t | R_BLE_GAP_ReadRemoteTransmitPowerLevel (uint16_t conn_hdl, uint8_t phy)<br>Read the transmit power level used by the remote device. More... |
| ble_status_t | R_BLE_GAP_SetPathLossReportingParam (st_ble_gap_set_path_loss_rpt_param_t *p_loss_rpt_param)<br>Set the path loss threshold reporting parameters for the ACL connection identified. More... |
| ble_status_t | R_BLE_GAP_SetPathLossReportingEnable (uint16_t conn_hdl, uint8_t enable)<br>Enable or disable path loss reporting for the ACL connection. More... |
| ble_status_t | R_BLE_GAP_SetTransmitPowerReportingEnable (uint16_t conn_hdl, uint8_t local_enable, uint8_t remote_enable)<br>Enable or disable the transmit power level changing report. More... |
| ble_status_t | R_BLE_GAP_SetDataRelatedAddrChanges (uint8_t adv_hdl, uint8_t change_reason)<br>Specifies circumstances when the Controller shall refresh any Resolvable Private Address. More... |
| ble_status_t | R_BLE_GAP_TestEnd (void)<br>Stop any test which is in progress. The Num_Packets for a transmitter test shall be reported as 0x0000. The Num_Packets is an unsigned number and contains the number of received packets. More... |
| ble_status_t | R_BLE_GAP_ReqPeerSCA (uint16_t conn_hdl)<br>Read the Sleep Clock Accuracy (SCA) of the peer device. More... |
| ble_status_t | R_BLE_GAP_EnhancedReadTxPowerLevel (uint16_t conn_hdl, uint8_t phy) |

| | |
|---|---|
| | Read the current and maximum transmit power levels of the local Controller on the ACL connection identified by the Connection_Handle parameter and the PHY indicated by the PHY parameter. More... |
| ble_status_t | R_BLE_GAP_SetHostFeat (uint8_t bit_number, uint8_t bit_value)<br>Set or clear a bit controlled by the Host in the Link Layer FeatureSet stored in the Controller. More... |

## Detailed Description

(end addtogroup BLE_API)

## Data Structures

| | |
|---|---|
| struct | st_ble_evt_data_t<br>st_ble_evt_data_t is the type of the data notified in a GAP Event. More... |
| struct | st_ble_dev_addr_t<br>st_ble_dev_addr_t is the type of bluetooth device address(BD_ADDR). More... |
| struct | st_ble_gap_ext_adv_param_t<br>Advertising parameters. More... |
| struct | st_ble_gap_adv_data_t<br>Advertising data/scan response data/periodic advertising data. More... |
| struct | st_ble_gap_perd_adv_param_t<br>Periodic advertising parameter. More... |
| struct | st_ble_gap_scan_phy_param_t<br>Scan parameters per scan PHY. More... |
| struct | st_ble_gap_ext_scan_param_t<br>Scan parameters. More... |

| | |
|---|---|
| struct | st_ble_gap_scan_on_t<br>Parameters configured when scanning starts. More... |
| struct | st_ble_gap_conn_param_t<br>Connection parameters included in connection interval, slave latency, supervision timeout, ce length. More... |
| struct | st_ble_gap_conn_phy_param_t<br>Connection parameters per PHY. More... |
| struct | st_ble_gap_create_conn_param_t<br>Connection parameters used in R_BLE_GAP_CreateConn(). More... |
| struct | st_ble_gap_rslv_list_key_set_t<br>IRK of a remote device and IRK type of local device used in R_BLE_GAP_ConfRslvList(). More... |
| struct | st_ble_gap_set_phy_param_t<br>PHY configuration parameters used in R_BLE_GAP_SetPhy(). More... |
| struct | st_ble_gap_set_def_phy_param_t<br>PHY preferences which allows a remote device to set used in R_BLE_GAP_SetDefPhy(). More... |
| struct | st_ble_gap_auth_info_t<br>Pairing parameters required from a remote device or information about keys distributed from a remote device. More... |
| struct | st_ble_gap_key_dist_t<br>Keys distributed from a remote device. More... |
| struct | st_ble_gap_key_ex_param_t<br>This structure includes the distributed keys and negotiated LTK size. More... |

| | |
|---|---|
| struct | st_ble_gap_pairing_param_t<br>Pairing parameters used in R_BLE_GAP_SetPairingParams(). More... |
| struct | st_ble_gap_oob_data_t<br>Oob data received from the remote device. This is used in R_BLE_GAP_SetRemOobData(). More... |
| struct | st_ble_gap_cte_antenna_info_t<br>This is the parameters used in R_BLE_GAP_GetAntennaInfo(). More... |
| struct | st_ble_gap_recv_test_param_t<br>This is the parameters used in R_BLE_GAP_ReceiverTest() More... |
| struct | st_ble_gap_trans_test_param_t<br>This is the parameters used in R_BLE_GAP_TransmitterTest(). More... |
| struct | st_ble_gap_set_path_loss_rpt_param_t<br>This is the parameters used in R_BLE_GAP_SetPathLossReportingParam(). More... |
| struct | st_ble_gap_cte_connless_t<br>connectionless CTE param More... |
| struct | st_ble_gap_cte_conn_t<br>connection CTE param More... |
| struct | st_ble_gap_cte_connless_recv_t<br>connectionless CTE receive param More... |
| struct | st_ble_gap_cte_conn_rx_param_t<br>connection CTE receive param More... |
| struct | st_ble_gap_cte_conn_req_t<br>connection CTE request More... |

| | |
|---|---|
| struct | st_ble_gap_subrate_param_t<br>subrating param More... |
| struct | st_ble_gap_ver_num_t<br>Version number of host stack. More... |
| struct | st_ble_gap_loc_ver_info_t<br>Version number of Controller. More... |
| struct | st_ble_gap_loc_dev_info_evt_t<br>Version information of local device. More... |
| struct | st_ble_gap_hw_err_evt_t<br>Hardware error that is notified from Controller. More... |
| struct | st_ble_gap_cmd_err_evt_t<br>HCI Command error. More... |
| struct | st_ble_gap_adv_rept_t<br>Advertising Report. More... |
| struct | st_ble_gap_ext_adv_rept_t<br>Extended Advertising Report. More... |
| struct | st_ble_gap_perd_adv_rept_t<br>Periodic Advertising Report. More... |
| struct | st_ble_gap_adv_rept_evt_t<br>Advertising report. More... |
| union | st_ble_gap_adv_rept_evt_t.param<br>Advertising Report. More... |

| | |
|---|---|
| struct | st_ble_gap_adv_set_evt_t<br>Advertising handle. More... |
| struct | st_ble_gap_adv_off_evt_t<br>Information about the advertising set which stops advertising. More... |
| struct | st_ble_gap_adv_data_evt_t<br>This structure notifies that advertising data has been set to Controller by R_BLE_GAP_SetAdvSresData(). More... |
| struct | st_ble_gap_rem_adv_set_evt_t<br>This structure notifies that an advertising set has been removed. More... |
| struct | st_ble_gap_conn_evt_t<br>This structure notifies that a link has been established. More... |
| struct | st_ble_gap_disconn_evt_t<br>This structure notifies that a link has been disconnected. More... |
| struct | st_ble_gap_rd_ch_map_evt_t<br>This structure notifies that Channel Map has been retrieved by R_BLE_GAP_ReadChMap(). More... |
| struct | st_ble_gap_rd_rssi_evt_t<br>This structure notifies that RSSI has been retrieved by R_BLE_GAP_ReadRssi(). More... |
| struct | st_ble_gap_dev_info_evt_t<br>This structure notifies that information about remote device has been retrieved by R_BLE_GAP_GetRemDevInfo(). More... |
| struct | st_ble_gap_conn_upd_evt_t<br>This structure notifies that connection parameters has been updated. More... |

| | |
|---|---|
| struct | st_ble_gap_conn_upd_req_evt_t<br>This structure notifies that a request for connection parameters update has been received. More... |
| struct | st_ble_gap_conn_hdl_evt_t<br>This structure notifies that a GAP Event that includes only connection handle has occurred. More... |
| struct | st_ble_gap_data_len_chg_evt_t<br>This structure notifies that the packet data length has been updated. More... |
| struct | st_ble_gap_rd_rpa_evt_t<br>This structure notifies that the local resolvable private address has been retrieved by R_BLE_GAP_ReadRpa(). More... |
| struct | st_ble_gap_phy_upd_evt_t<br>This structure notifies that PHY for a connection has been updated. More... |
| struct | st_ble_gap_phy_rd_evt_t<br>This structure notifies that the PHY settings has been retrieved by R_BLE_GAP_ReadPhy(). More... |
| struct | st_ble_gap_scan_req_recv_evt_t<br>This structure notifies that a Scan Request packet has been received from a Scanner. More... |
| struct | st_ble_gap_sync_est_evt_t<br>This structure notifies that a Periodic sync has been established. More... |
| struct | st_ble_gap_sync_hdl_evt_t<br>This structure notifies that a GAP Event that includes only sync handle has occurred. More... |
| struct | st_ble_gap_white_list_conf_evt_t |

| | |
|---|---|
| | This structure notifies that White List has been configured. More... |
| struct | st_ble_gap_rslv_list_conf_evt_t<br>This structure notifies that Resolving List has been configured. More... |
| struct | st_ble_gap_perd_list_conf_evt_t<br>This structure notifies that Periodic Advertiser List has been configured. More... |
| struct | st_ble_gap_set_priv_mode_evt_t<br>This structure notifies that Privacy Mode has been configured. More... |
| struct | st_ble_gap_pairing_req_evt_t<br>This structure notifies that a pairing request from a remote device has been received. More... |
| struct | st_ble_gap_passkey_display_evt_t<br>This structure notifies that a request for Passkey display in pairing has been received. More... |
| struct | st_ble_gap_num_comp_evt_t<br>This structure notifies that a request for Numeric Comparison in pairing has been received. More... |
| struct | st_ble_gap_key_press_ntf_evt_t<br>This structure notifies that the remote device has input a key in Passkey Entry. More... |
| struct | st_ble_gap_pairing_info_evt_t<br>This structure notifies that the pairing has completed. More... |
| struct | st_ble_gap_enc_chg_evt_t<br>This structure notifies that the encryption status of a link has been changed. More... |

| | |
|---|---|
| struct | st_ble_gap_peer_key_info_evt_t<br>This structure notifies that the remote device has distributed the keys. More... |
| struct | st_ble_gap_ltk_req_evt_t<br>This structure notifies that a LTK request from a remote device has been received. More... |
| struct | st_ble_gap_ltk_rsp_evt_t<br>This structure notifies that local device has replied to the LTK request from the remote device. More... |
| struct | st_ble_gap_sc_oob_data_evt_t<br>This structure notifies that OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData(). More... |
| struct | st_ble_gap_bond_info_t<br>Bonding information used in R_BLE_GAP_SetBondInfo(). More... |
| struct | st_cte_iq_sample_t<br>CTE IQ sample data. More... |
| struct | st_ble_gap_cte_connless_rept_t<br>connectionless CTE data report More... |
| struct | st_ble_gap_cte_conn_rept_t<br>connection CTE data report More... |
| struct | st_ble_subrate_upd_t<br>subrating update event More... |
| struct | st_ble_gap_past_est_evt_t<br>This structure notifies that. More... |
| struct | st_ble_gap_tx_power_reporting_evt_t |

| | |
|---|---|
| | This structure notifies that. More... |
| struct | st_ble_gap_pass_loss_thr_evt_t<br>This structure notifies that a path loss report has been received. More... |
| struct | st_ble_gap_req_peer_sca_evt_t<br>This structure notifies that a SCA request to a remote device has been completed. More... |
| struct | st_ble_gap_dtm_test_end_evt_t<br>report of dtm transmit/receive test end More... |
| struct | st_ble_gap_enhanced_read_tx_power_level_evt_t<br>Power level report of remove device. More... |

## Macros

| | |
|---|---|
| #define | BLE_BD_ADDR_LEN |
| #define | BLE_MASTER |
| #define | BLE_SLAVE |
| #define | BLE_GAP_ADDR_PUBLIC |
| #define | BLE_GAP_ADDR_RAND |
| #define | BLE_GAP_ADDR_RPA_ID_PUBLIC<br>Resolvable Private Address. More... |
| #define | BLE_GAP_ADDR_RPA_ID_RANDOM<br>Resolvable Private Address. More... |
| #define | BLE_GAP_AD_FLAGS_LE_LIM_DISC_MODE<br>LE Limited Discoverable Mode flag used in AD type. |
| #define | BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE |

| | |
|---|---|
| | LE General Discoverable Mode flag used in AD type. |
| #define | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED<br>BR/EDR Not Supported flag used in AD type. |
| #define | BLE_GAP_ADV_DATA_MODE<br>Advertising data. |
| #define | BLE_GAP_SCAN_RSP_DATA_MODE<br>Scan response data. |
| #define | BLE_GAP_PERD_ADV_DATA_MODE<br>Periodic advertising data. |
| #define | BLE_GAP_ADV_CH_37<br>Use 37 CH. |
| #define | BLE_GAP_ADV_CH_38<br>Use 38 CH. |
| #define | BLE_GAP_ADV_CH_39<br>Use 39 CH. |
| #define | BLE_GAP_ADV_CH_ALL<br>Use 37 - 39 CH. |
| #define | BLE_GAP_SCAN_PASSIVE<br>Passive Scan. |
| #define | BLE_GAP_SCAN_ACTIVE<br>Active Scan. |
| #define | BLE_GAP_SCAN_INTV_MIN<br>Active Scan. |

| | |
|---|---|
| #define | BLE_GAP_SCAN_FILT_DUPLIC_DISABLE<br>Duplicate filter disabled. |
| #define | BLE_GAP_SCAN_FILT_DUPLIC_ENABLE<br>Duplicate filter enabled. |
| #define | BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD<br>Duplicate filtering enabled, reset for each scan period. |
| #define | BLE_GAP_SCAN_ALLOW_ADV_ALL<br>Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device. |
| #define | BLE_GAP_SCAN_ALLOW_ADV_WLST<br>Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not addressed to local device is ignored. |
| #define | BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED<br>Accept all advertising and scan response PDUs except directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted. |
| #define | BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST<br>Accept all advertising and scan response PDUs.<br>The following are excluded. More... |
| #define | BLE_GAP_INIT_FILT_USE_ADDR<br>White List is not used. |
| #define | BLE_GAP_INIT_FILT_USE_WLST<br>White List is used. |
| #define | BLE_GAP_DATA_0_CLEAR |

| | |
|---|---|
| | Clear the advertising data/scan response data/periodic advertising data in the advertising set. |
| #define | BLE_GAP_DATA_0_DID_UPD<br>Update Advertising DID without changing advertising data. |
| #define | BLE_GAP_NET_PRIV_MODE<br>Network Privacy Mode. |
| #define | BLE_GAP_DEV_PRIV_MODE<br>Device Privacy Mode. |
| #define | BLE_GAP_REM_FEATURE_SIZE<br>The length of the features supported by a remote device. |
| #define | BLE_GAP_NOT_AUTHORIZED<br>Not authorize the remote device. |
| #define | BLE_GAP_AUTHORIZED<br>Authorize the remote device. |
| #define | BLE_GAP_RMV_ADV_SET_REM_OP<br>Delete an advertising set. |
| #define | BLE_GAP_RMV_ADV_SET_CLR_OP<br>Delete all the advertising sets. |
| #define | BLE_GAP_SC_PROC_GEN<br>General Discovery Procedure. |
| #define | BLE_GAP_SC_PROC_LIM<br>Limited Discovery Procedure. |
| #define | BLE_GAP_SC_PROC_OBS |

| | |
|---|---|
| | Observation Procedure. |
| #define | BLE_GAP_LIST_ADD_DEV<br>Add the device to the list. |
| #define | BLE_GAP_LIST_REM_DEV<br>Delete the device from the list. |
| #define | BLE_GAP_LIST_CLR<br>Clear the list. |
| #define | BLE_GAP_WHITE_LIST_MAX_ENTRY<br>The maximum entry number of White List. |
| #define | BLE_GAP_RSLV_LIST_MAX_ENTRY<br>The maximum entry number of Resolving List. |
| #define | BLE_GAP_PERD_LIST_MAX_ENTRY<br>The maximum entry number of Periodic Advertiser List. |
| #define | BLE_GAP_RPA_DISABLED<br>Disable RPA generation/resolution. |
| #define | BLE_GAP_RPA_ENABLED<br>Enable RPA generation/resolution. |
| #define | BLE_GAP_RL_LOC_KEY_ALL_ZERO<br>All-zero IRK. |
| #define | BLE_GAP_RL_LOC_KEY_REGISTERED<br>The IRK registered by R_BLE_GAP_SetLocIdInfo(). |
| #define | BLE_MAX_NO_OF_ADV_SETS_SUPPORTED<br>The maximum number of advertising set for the Abstraction API. |

| | |
|---|---|
| #define | BLE_GAP_LEGACY_PROP_ADV_IND<br>Connectable and scannable undirected Legacy Advertising Packet. |
| #define | BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND<br>Connectable directed (low duty cycle) Legacy Advertising Packet. |
| #define | BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND<br>Connectable directed (high duty cycle) Legacy Advertising Packet. |
| #define | BLE_GAP_LEGACY_PROP_ADV_SCAN_IND<br>Scannable undirected Legacy Advertising Packet. |
| #define | BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND<br>Non-connectable and non-scannable undirected Legacy Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT<br>Connectable and non-scannable undirected Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT<br>Connectable and non-scannable directed (low duty cycle) Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_HDC_DIRECT<br>Connectable and non-scannable directed (high duty cycle) Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT<br>Non-connectable and scannable undirected Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT<br>Non-connectable and scannable directed (low duty cycle) Extended Advertising Packet. |

| | |
|---|---|
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT<br>Non-connectable and scannable directed (high duty cycle) Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT<br>Non-connectable and non-scannable undirected Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT<br>Non-connectable and non-scannable directed (low duty cycle) Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT<br>Non-connectable and non-scannable directed (high duty cycle) Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_ANONYMOUS<br>Omit the advertiser address from Extended Advertising Packet. |
| #define | BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER<br>Indicate that the advertising data includes TX Power. |
| #define | BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY<br>Process scan and connection requests from all devices. |
| #define | BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_ANY<br>Process connection requests from all devices and scan requests from only devices that are in the White List. |
| #define | BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_WLST<br>Process scan requests from all devices and connection requests from only devices that are in the White List. |
| #define | BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_WLST<br>Process scan and connection requests from only devices in the White List. |

| | |
|---|---|
| #define | BLE_GAP_ADV_PHY_1M<br>Use 1M PHY. |
| #define | BLE_GAP_ADV_PHY_2M<br>Use 2M PHY. |
| #define | BLE_GAP_ADV_PHY_CD<br>Use Coded PHY. |
| #define | BLE_GAP_SCAN_REQ_NTF_DISABLE<br>Disable Scan Request Notification. |
| #define | BLE_GAP_SCAN_REQ_NTF_ENABLE<br>Enable Scan Request Notification. |
| #define | BLE_GAP_PERD_PROP_TX_POWER<br>Indicate that periodic advertising data includes Tx Power. |
| #define | BLE_GAP_INVALID_ADV_HDL<br>Invalid advertising handle. |
| #define | BLE_GAP_SET_PHYS_HOST_PREF_1M<br>Use 1M PHY. |
| #define | BLE_GAP_SET_PHYS_HOST_PREF_2M<br>Use 2M PHY. |
| #define | BLE_GAP_SET_PHYS_HOST_PREF_CD<br>Use Coded PHY. |
| #define | BLE_GAP_SET_PHYS_OP_HOST_NO_PREF<br>No preferred coding. |

| | |
|---|---|
| #define | BLE_GAP_SET_PHYS_OP_HOST_PREF_S_2<br>Use S=2 coding. |
| #define | BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8<br>Use S=8 coding. |
| #define | BLE_GAP_CONN_UPD_MODE_REQ<br>Request for updating the connection parameters. |
| #define | BLE_GAP_CONN_UPD_MODE_RSP<br>Reply a connection parameter update request. |
| #define | BLE_GAP_CONN_UPD_ACCEPT<br>Accept the update request. |
| #define | BLE_GAP_CONN_UPD_REJECT<br>Reject the update request. |
| #define | BLE_GAP_CH_MAP_SIZE<br>The size of channel map. |
| #define | BLE_GAP_INVALID_CONN_HDL<br>Invalid Connection handle. |
| #define | BLE_GAP_NOT_USE_CONN_HDL<br>This macro indicates that connection handle is not used. |
| #define | BLE_GAP_INIT_CONN_HDL<br>Initial Connection handle. |
| #define | BLE_GAP_PAIRING_ACCEPT<br>Accept a request regarding pairing. |
| #define | BLE_GAP_PAIRING_REJECT |

| | |
|---|---|
| | Reject a request regarding pairing. |
| #define | BLE_GAP_LTK_REQ_ACCEPT<br>Reply for the LTK request. |
| #define | BLE_GAP_LTK_REQ_DENY<br>Reject the LTK request. |
| #define | BLE_GAP_LESC_PASSKEY_ENTRY_STARTED<br>Notify that passkey entry started. |
| #define | BLE_GAP_LESC_PASSKEY_DIGIT_ENTERED<br>Notify that passkey digit entered. |
| #define | BLE_GAP_LESC_PASSKEY_DIGIT_ERASED<br>Notify that passkey digit erased. |
| #define | BLE_GAP_LESC_PASSKEY_CLEARED<br>Notify that passkey cleared. |
| #define | BLE_GAP_LESC_PASSKEY_ENTRY_COMPLETED<br>Notify that passkey entry completed. |
| #define | BLE_GAP_SEC_MITM_BEST_EFFORT<br>MITM Protection not required. |
| #define | BLE_GAP_SEC_MITM_STRICT<br>MITM Protection required. |
| #define | BLE_GAP_KEY_DIST_ENCKEY<br>LTK. |
| #define | BLE_GAP_KEY_DIST_IDKEY<br>IRK and Identity Address. |

| | |
|---|---|
| #define | BLE_GAP_KEY_DIST_SIGNKEY<br>CSRK. |
| #define | BLE_GAP_ID_ADDR_SIZE<br>The size of identity address. |
| #define | BLE_GAP_IRK_SIZE<br>The size of IRK. |
| #define | BLE_GAP_CSRK_SIZE<br>The size of CSRK. |
| #define | BLE_GAP_LTK_SIZE<br>The size of LTK. |
| #define | BLE_GAP_EDIV_SIZE<br>The size of EDIV. |
| #define | BLE_GAP_RAND_64_BIT_SIZE<br>The size of Rand. |
| #define | BLE_GAP_UNAUTH_PAIRING<br>Unauthenticated pairing. |
| #define | BLE_GAP_AUTH_PAIRING<br>Authenticated pairing. |
| #define | BLE_GAP_LEGACY_PAIRING<br>Legacy pairing. |
| #define | BLE_GAP_LESC_PAIRING<br>Secure Connections. |

| | |
|---|---|
| #define | BLE_GAP_BONDING_NONE<br>The device doesn't support Bonding. |
| #define | BLE_GAP_BONDING<br>The device supports Bonding. |
| #define | BLE_GAP_IOCAP_DISPLAY_ONLY<br>Display Only iocapability. More... |
| #define | BLE_GAP_IOCAP_DISPLAY_YESNO<br>Display Yes/No iocapability. More... |
| #define | BLE_GAP_IOCAP_KEYBOARD_ONLY<br>Keyboard Only iocapability. More... |
| #define | BLE_GAP_IOCAP_NOINPUT_NOOUTPUT<br>No Input No Output iocapability. More... |
| #define | BLE_GAP_IOCAP_KEYBOARD_DISPLAY<br>Keyboard Display iocapability. More... |
| #define | BLE_GAP_OOB_DATA_NOT_PRESENT<br>Reply that No OOB data has been received when pairing. |
| #define | BLE_GAP_OOB_DATA_PRESENT<br>Reply that the OOB data has been received when pairing. |
| #define | BLE_GAP_SC_BEST_EFFORT<br>Accept Legacy pairing and Secure Connections. |
| #define | BLE_GAP_SC_STRICT<br>Accept only Secure Connections. |
| #define | BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT |

| | |
|---|---|
| | Not support for Key Press Notification. |
| #define | BLE_GAP_SC_KEY_PRESS_NTF_SPRT<br>Support for Key Press Notification. |
| #define | BLE_GAP_LEGACY_OOB_SIZE<br>The size of Temporary Key for OOB in legacy pairing. |
| #define | BLE_GAP_OOB_CONFIRM_VAL_SIZE<br>The size of Confirmation Value for OOB in Secure Connections. |
| #define | BLE_GAP_OOB_RANDOM_VAL_SIZE<br>The size of Rand for OOB in Secure Connections. |
| #define | BLE_GAP_SEC_DEL_LOC_NONE<br>Delete no local keys. |
| #define | BLE_GAP_SEC_DEL_LOC_IRK<br>Delete local IRK. |
| #define | BLE_GAP_SEC_DEL_LOC_CSRK<br>Delete local CSRK. |
| #define | BLE_GAP_SEC_DEL_LOC_ALL<br>Delete all local keys. |
| #define | BLE_GAP_SEC_DEL_REM_NONE<br>Delete no remote device keys. |
| #define | BLE_GAP_SEC_DEL_REM_SA<br>Delete a key specified by the p_addr parameter. |
| #define | BLE_GAP_SEC_DEL_REM_NOT_CONN<br>Delete keys of not connected remote devices. |

| | |
|---|---|
| #define | BLE_GAP_SEC_DEL_REM_ALL<br>Delete all remote device keys. |
| #define | BLE_GAP_CTE_DISABLED<br>Disable CTE transmission or receive. |
| #define | BLE_GAP_CTE_ENABLED<br>Enable CTE transmission or receive. |
| #define | BLE_GAP_CTE_MAX_ANTENNA<br>Max antenna number. |
| #define | BLE_GAP_CTE_TYPE_AOA<br>CTE type AoA. |
| #define | BLE_GAP_CTE_TYPE_AOD_1US<br>CTE type AoD with slot of 1 us. |
| #define | BLE_GAP_CTE_TYPE_AOD_2US<br>CTE type AoD with slot of 2 us. |

## Typedefs

| | |
|---|---|
| typedef void(* | ble_gap_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data)<br>ble_gap_app_cb_t is the GAP Event callback function type. More... |
| typedef void(* | ble_gap_del_bond_cb_t) (st_ble_dev_addr_t *p_addr)<br>ble_gap_del_bond_cb_t is the type of the callback function for delete bonding information stored in non-volatile area.<br>This type is used in R_BLE_GAP_DeleteBondInfo(). More... |
| typedef st_ble_gap_ext_adv_param_t | st_ble_gap_adv_param_t<br>Advertising parameters. More... |

| | |
|---|---|
| typedef st_ble_gap_ext_scan_param_t | st_ble_gap_scan_param_t<br><br>Scan parameters. More... |

### Enumerations

| | |
|---|---|
| enum | e_ble_gap_evt_t<br><br>GAP Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_evt_data_t

struct st_ble_evt_data_t

st_ble_evt_data_t is the type of the data notified in a GAP Event.

| Data Fields | | |
|---|---|---|
| uint16_t | param_len | The size of GAP Event parameters. |
| void * | p_param | GAP Event parameters. This parameter differs in each GAP Event. |

### ◆ st_ble_dev_addr_t

struct st_ble_dev_addr_t

st_ble_dev_addr_t is the type of bluetooth device address(BD_ADDR).

***Note***

*The BD address setting format is little endian.*
*If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.*

| Data Fields | | |
|---|---|---|
| uint8_t | addr[BLE_BD_ADDR_LEN] | BD_ADDR. |
| uint8_t | type | Bluetooth address type.<br>macro: description<br>BLE_GAP_ADDR_PUBLIC(0x00): Public Address.<br>BLE_GAP_ADDR_RAND(0x01): Random Address. |

### ◆ st_ble_gap_ext_adv_param_t

struct st_ble_gap_ext_adv_param_t

Advertising parameters.

Data Fields

| | | |
|---|---|---|
| uint8_t | adv_hdl | Advertising handle identifying the advertising set to be set the advertising parameters.<br><br>Valid range is 0x00 - 0x03.<br>In the first advertising parameters setting, the advertising set specified by adv_hdl is generated.<br>The Advertising Set ID(Advertising SID) of the advertising set is same as adv_hdl. |
| uint16_t | adv_prop_type | Advertising packet type.<br><br>Legacy advertising PDU type, or bitwise or of Extended advertising PDU type and Extended advertising option.<br><br>category: Legacy Advertising PDU type<br>macro: BLE_GAP_LEGACY_PROP_ADV_IND(0x0013) — Connectable and scannable undirected Legacy Advertising Packet<br>macro: BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND(0x0015) — Connectable directed (low duty cycle) Legacy Advertising Packet<br>macro: BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND(0x001D) — Connectable directed (high duty cycle) Legacy Advertisin |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | g Packet |
| | | | | BLE_GAP_LEGACY_PROP_ADV_SCAN_IND(0x0012) | Scannable undirected Legacy Advertising Packet |
| | | | | BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND(0x0010) | Non-connectable and non-scannable undirected Legacy Advertising Packet |
| | | | Extended Advertising PDU type | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT(0x0001) | Connectable and non-scannable undirected Extended Advertising Packet |
| | | | | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT(0x0005) | Connectable and non-scannable directed (low duty cycle) Extended Advertising Packet |
| | | | | BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_HDC_DIRECT(0x000D) | Connectable and non-scannable directed (high duty cycle) Extended Advertising Packet |
| | | | | BLE_GAP_EXT_PR | Non-connectable |

| | | | | |
|---|---|---|---|---|
| | | | OP_ADV_NOCONN_SCAN_UNDIRECT(0x0002) | and scannable undirected Extended Advertising Packet |
| | | | BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT(0x0006) | Non-connectable and scannable directed (low duty cycle) Extended Advertising Packet |
| | | | BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT(0x000E) | Non-connectable and scannable directed (high duty cycle) Extended Advertising Packet |
| | | | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT(0x0000) | Non-connectable and non-scannable undirected Extended Advertising Packet |
| | | | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT(0x0004) | Non-connectable and non-scannable directed (low duty cycle) Extended Adverti |

| | | | | |
|---|---|---|---|---|
| | | | | sing Packet |
| | | | BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT(0x000C) | Non-connectable and non-scannable directed (high duty cycle) Extended Advertising Packet |
| | | Extended Advertising Option | BLE_GAP_EXT_PROP_ADV_ANONYMOUS(0x0020) | Omit the advertiser address from Extended Advertising Packet. |
| | | | BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER(0x0040) | Indicate that the advertising data includes TX Power. |
| uint32_t | adv_intv_min | Minimum advertising interval.<br><br>Time(ms) = adv_intv_min * 0.625.<br>Valid range is 0x00000020 - 0x00FFFFFF. | | |
| uint32_t | adv_intv_max | Maximum Advertising interval.<br><br>Time(ms) = adv_intv_max * 0.625.<br>Valid range is 0x00000020 - 0x00FFFFFF. | | |
| uint8_t | adv_ch_map | The adv_ch_map is channels used in advertising with primary advertising channels.<br><br>It is a bitwise OR of the following values.<br><br>macro \| description | | |

| | | |
|---|---|---|
| | | BLE_GAP_ADV_CH_37(0x01) Use 37 CH.<br>BLE_GAP_ADV_CH_38(0x02) Use 38 CH.<br>BLE_GAP_ADV_CH_39(0x04) Use 39 CH.<br>BLE_GAP_ADV_CH_ALL(0x07) Use 37 - 39 CH. |
| uint8_t | o_addr_type | Own BD Address Type.<br>BLE_GAP_ADDR_PUBLIC(0x00) Public Address<br>BLE_GAP_ADDR_RAND(0x01) Random Address<br>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used.<br>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, the random address specified by the o_addr field is used. |
| uint8_t | o_addr[BLE_BD_ADDR_LEN] | Random address set to the advertising set, when the o_addr_type field is BLE_GAP_ADDR_RAND.<br><br>When the o_addr_type field is other than BLE_GAP_ADDR_RAND, this field |

<table>
<tr><td></td><td></td><td>is ignored.<br><br>Note<br>The BD address setting format is little endian.<br>If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.</td></tr>
<tr><td>uint8_t</td><td>p_addr_type</td><td>Peer address type.<br><br>When the Advertising PDU type is other than directed or the o_addr_type is BLE_GAP_ADDR_PUBLIC or BLE_GAP_ADDR_RAND,this field is ignored.<br><br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>p_addr[BLE_BD_ADDR_LEN]</td><td>Peer address.<br><br>When the Advertising PDU type is other than directed or the o_addr_type is BLE_GAP_ADDR_PUBLIC or BLE_GAP_ADDR_RAND,this field is ignored.<br><br>Note<br>The BD address setting format is little endian.<br>If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.</td></tr>
<tr><td>uint8_t</td><td>filter_policy</td><td>Advertising Filter Policy.<br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| | | BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_ANY(0x01) Process connection requests from all devices and scan requests from only devices that are in the White List.<br>BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_WLST(0x02) Process scan requests from all devices and connection requests from only devices that are in the White List.<br>BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_WLST(0x03) Process scan and connection requests from only devices in the White List. |
| uint8_t | adv_phy | Primary ADV PHY.<br><br>In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.<br><br>macro / description<br>BLE_GAP_ADV_PHY_1M(0x01) Use 1M PHY as Primary Advertising PHY. When the adv_prop_type field is Legacy Advertising PDU type, this field shall be set to BLE_GAP_ADV_PHY_1M.<br>BLE_GAP_ADV_PHY_CD(0x03) Use Coded PHY(S=8) as Primary Advertising |

<table>
<tr><td></td><td></td><td>PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme().</td></tr>
<tr><td>uint8_t</td><td>sec_adv_max_skip</td><td>Secondary ADV Max Skip.<br><br>Valid range is 0x00 - 0xFF.<br>When this field is 0x00, AUX_ADV_IND is sent before the next advertising event.<br>When the adv_prop_type field is Legacy Advertising PDU, this field is ignored.</td></tr>
<tr><td>uint8_t</td><td>sec_adv_phy</td><td>Secondary ADV Phy.<br><br>When the adv_prop_type is Legacy Advertising PDU, this field is ignored.<br><br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Secondary Advertising PHY.</td></tr><tr><td>BLE_GAP_ADV_PHY_2M(0x02)</td><td>Use 2M PHY as Secondary Advertising PHY.</td></tr><tr><td>BLE_GAP_ADV_PHY_CD(0x03)</td><td>Use Coded PHY(S=8) as Secondary Advertising PHY.</td></tr></table>Coding scheme is configured by R_BLE_VS_SetCodingScheme().</td></tr>
<tr><td>uint8_t</td><td>scan_req_ntf_flag</td><td>Scan Request Notifications Flag.<br><br>When the adv_prop_type field is non-scannable Advertising PDU, this field is ignored.<br><br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SCAN_REQ_NTF_DISABLE(0x00)</td><td>Disable Scan Request Notification.</td></tr><tr><td>BLE_GAP_SCA</td><td>Enable Scan</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| | | N_REQ_NTF_ENABLE(0x01) Request Notification. When a Scan Request Packet from Scanner has been received, the BLE_GAP_EVENT_SCAN_REQ_RECV event is notified. |

### ◆ st_ble_gap_adv_data_t

| struct st_ble_gap_adv_data_t | | |
|---|---|---|
| Advertising data/scan response data/periodic advertising data. | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set to be set advertising data/scan response/periodic advertising data.<br><br>Valid range is 0x00 - 0x03. |
| uint8_t | data_type | Data type.<br>macro: description<br>BLE_GAP_ADV_DATA_MODE(0x00): Advertising data.<br>BLE_GAP_SCAN_RSP_DATA_MODE(0x01): Scan response data.<br>BLE_GAP_PERD_ADV_DATA_MODE(0x02): Periodic advertising data. |
| uint16_t | data_length | The length of advertising data/scan response data/periodic advertising data (in bytes).<br><br>In case of Legacy Advertising PDU, the length is 0 - 31 bytes.<br>In case of Extended Advertising PDU, the length is 0 - 1650 bytes.<br>Note that the length of the advertising data/scan response data in the BLE_MAX_NO_OF_A |

<table>
<tr><td></td><td></td><td>DV_SETS_SUPPORTED number of the advertising sets may not exceed the buffer size(4250 bytes) in Controller.<br><br>In case of periodic advertising data, the length is 0 - 1650 bytes.<br>Note that the length of the periodic advertising data in the BLE_MAX_NO_OF_ADV_SETS_SUPPORTED number of the advertising sets may not exceed the buffer size(4306 bytes) in Controller.<br><br>When this field is 0, the operations specified by the zero_length_flag is executed.</td></tr>
<tr><td>uint8_t *</td><td>p_data</td><td>Advertising data/scan response data/periodic advertising data.<br><br>When the data_length field is 0, this field is ignored.</td></tr>
<tr><td>uint8_t</td><td>zero_length_flag</td><td>Operation when the data_length field is 0.<br><br>If the data_length is other than 0, this field is ignored.<br><br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_DATA_0_CLEAR(0x01)</td><td>Clear the advertising data/scan response data/periodic advertising data in the advertising set.</td></tr><tr><td>BLE_GAP_DATA_0_DID_UPD(0x02)</td><td>Update Advertising DID without changing advertising data. If the data_type field is BLE_GAP_ADV_DATA_MODE, this value is allowed.</td></tr></table></td></tr>
</table>

### ◆ st_ble_gap_perd_adv_param_t

| struct st_ble_gap_perd_adv_param_t | | |
|---|---|---|
| Periodic advertising parameter. | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set to be set periodic advertising parameter.<br><br>Valid range is 0x00 - 0x03. |
| uint16_t | prop_type | Periodic ADV Properties.<br><br>The prop_type field is set to the following values.<br>If the type of the periodic advertising data cannot be applied from the following, set 0x0000.<br><br>macro: BLE_GAP_PERD_PROP_TX_POWER(0x0040) — description: Indicate that periodic advertising data includes Tx Power. |
| uint16_t | perd_intv_min | Minimum Periodic Advertising Interval.<br><br>Time(ms) = perd_intv_min * 1.25.<br>Valid range is 0x0006 - 0xFFFF. |
| uint16_t | perd_intv_max | Maximum Periodic Advertising Interval.<br><br>Time(ms) = perd_intv_max * 1.25.<br>Valid range is 0x0006 - 0xFFFF. |

### ◆ st_ble_gap_scan_phy_param_t

| struct st_ble_gap_scan_phy_param_t | | |
|---|---|---|
| Scan parameters per scan PHY.<br><br>In case of start scanning with both 1M PHY and Coded PHY, adjust scan windows and scan intervals according to the following.<br>p_phy_param_1M->scan_window / p_phy_param_1M->scan_intv + p_phy_param_coded->scan_window / p_phy_param_coded->scan_intv <= 1 | | |
| Data Fields | | |
| uint8_t | scan_type | Scan type. |

| | | |
|---|---|---|
| | | macro / description<br>BLE_GAP_SCAN_PASSIVE(0x00) — Passive Scan.<br>BLE_GAP_SCAN_ACTIVE(0x01) — Active Scan. |
| uint16_t | scan_intv | Scan interval.<br><br>interval(ms) = scan_intv * 0.625.<br>Valid range is 0x0000 and 0x0004 - 0xFFFF. |
| uint16_t | scan_window | Scan window.<br><br>window(ms) = scan_window * 0.625.<br>Valid range is 0x0000 and 0x0004 - 0xFFFF. |

### ◆ st_ble_gap_ext_scan_param_t

| struct st_ble_gap_ext_scan_param_t | | |
|---|---|---|
| Scan parameters. | | |
| Data Fields | | |
| uint8_t | o_addr_type | Own BD Address Type.<br><br>In case of passive scan, this field is ignored.<br><br>macro / description<br>BLE_GAP_ADDR_PUBLIC(0x00) — Public Address<br>BLE_GAP_ADDR_RAND(0x01) — Random Address<br>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02) — Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used. |

| | | |
|---|---|---|
| | | BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, the random address set by R_BLE_GAP_SetRandAddr() is used. |
| uint8_t | filter_policy | Scan Filter Policy.<br>macro / description<br>BLE_GAP_SCAN_ALLOW_ADV_ALL(0x00) Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device.<br>BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01) Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not addressed to local device is ignored.<br>BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED(0x02) Accept all advertising and scan response PDUs except directed advertising PDUs whose |

| | | | | |
|---|---|---|---|---|
| | | | | the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted. |
| | | | BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST(0x03) | Accept all advertising and scan response PDUs. The following are excluded:<br><br>Advertising and scan response PDUs where the advertiser's identity address is not in the White List.<br><br>Directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the |

| | | |
|---|---|---|
| | | local resolvable private address are accepted. |
| st_ble_gap_scan_phy_param_t * | p_phy_param_1M | Scan parameters 1M PHY.<br><br>When this field is NULL, Controller doesn't set the scan parameters for 1M PHY. |
| st_ble_gap_scan_phy_param_t * | p_phy_param_coded | Scan parameters Coded PHY.<br><br>When this field is NULL, Controller doesn't set the scan parameters for Coded PHY. |

### ◆ st_ble_gap_scan_on_t

| struct st_ble_gap_scan_on_t | | |
|---|---|---|
| Parameters configured when scanning starts. | | |
| Data Fields | | |
| uint8_t | proc_type | Procedure type.<br>macro: description<br>BLE_GAP_SC_PROC_OBS(0x00): Observation Procedure. Notify all advertising PDUs.<br>BLE_GAP_SC_PROC_LIM(0x01): Limited Discovery Procedure. Notify advertising PDUs from only devices in the limited discoverable mode.<br>BLE_GAP_SC_PROC_GEN(0x02): General Discovery Procedure. Notify advertising PDUs from devices in the limited discoverable mode and the general discoverable |

| | | |
|---|---|---|
| | | mode. |
| uint8_t | filter_dups | Filter duplicates.<br>macro / description<br>BLE_GAP_SCAN_FILT_DUPLIC_DISABLE(0x00) — Duplicate filter disabled.<br>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE(0x01) — Duplicate filter enabled.<br>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02) — Duplicate filtering enabled, reset for each scan period |
| uint16_t | duration | Scan duration.<br><br>Time(ms) = duration * 10.<br>Valid range is 0x0000 - 0xFFFF.<br>If this field is set to 0x0000, scanning is continued until R_BLE_GAP_StopScan() is called.<br>When the period field is zero and the time specified the duration field expires, BLE_GAP_EVENT_SCAN_TO event notifies the application layer that scanning stops. |
| uint16_t | period | Scan period.<br><br>Time(s) = N * 1.28.<br>Valid range is 0x0000 - 0xFFFF.<br>If the duration field is set to 0x0000, this field is ignored. |

### ◆ st_ble_gap_conn_param_t

| struct st_ble_gap_conn_param_t |
|---|
| Connection parameters included in connection interval, slave latency, supervision timeout, ce length.<br><br>This structure is used in R_BLE_GAP_CreateConn() and R_BLE_GAP_UpdConn().<br><br>Set the fields in this structure to match the following condition.<br><br>Supervision_timeout(ms) >= (1 + conn_latency) * conn_intv_max_Time(ms)<br><br>conn_intv_max_Time(ms) = conn_intv_max * 1.25 Supervision_timeout(ms) = sup_to * 10 |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | conn_intv_min | Minimum connection interval.<br><br>Time(ms) = conn_intv_min * 1.25.<br>Valid range is 0x0006 - 0x0C80. |
| uint16_t | conn_intv_max | Maximum connection interval.<br><br>Time(ms) = conn_intv_max * 1.25.<br>Valid range is 0x0006 - 0x0C80. |
| uint16_t | conn_latency | Slave latency.<br><br>Valid range is 0x0000 - 0x01F3. |
| uint16_t | sup_to | Supervision timeout.<br><br>Time(ms) = sup_to * 10.<br>Valid range is 0x000A - 0x0C80. |
| uint16_t | min_ce_length | Minimum CE Length.<br><br>Valid range is 0x0000 - 0xFFFF. |
| uint16_t | max_ce_length | Maximum CE Length.<br><br>Valid range is 0x0000 - 0xFFFF. |

### ◆ st_ble_gap_conn_phy_param_t

| struct st_ble_gap_conn_phy_param_t | | |
|---|---|---|
| Connection parameters per PHY. | | |
| Data Fields | | |
| uint16_t | scan_intv | Scan interval.<br><br>Time(ms) = scan_intv * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| uint16_t | scan_window | Scan window.<br><br>Time(ms) = scan_window * 0.625.<br>Valid range is 0x0004 - 0xFFFF. |
| st_ble_gap_conn_param_t * | p_conn_param | Connection interval, slave latency, supervision timeout, and CE length. |

### ◆ st_ble_gap_create_conn_param_t

| struct st_ble_gap_create_conn_param_t |
|---|

Connection parameters used in R_BLE_GAP_CreateConn().

| Data Fields | | |
|---|---|---|
| uint8_t | init_filter_policy | This field specifies whether the White List is used or not, when connecting with a remote device.<br>macro: BLE_GAP_INIT_FILT_USE_ADDR(0x00) — description: White List is not used. The remote device to be connected is specified by the *remote_bd_addr* field and the *remote_bd_addr_type* field is used.<br>macro: BLE_GAP_INIT_FILT_USE_WLST(0x01) — description: White List is used. The remote device registered in White List is connected with local device. The *remote_bd_addr* field and the *remote_bd_addr_type* field are ignored. |
| uint8_t | remote_bd_addr[BLE_BD_ADDR_LEN] | Address of the device to be connected.<br>*Note*<br>*The BD address setting format is little endian. If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.* |
| uint8_t | remote_bd_addr_type | Address type of the device to be connected.<br>macro \| description |

| | | |
|---|---|---|
| | | BLE_GAP_ADDR_PUBLIC(0x00) Public Address or Public Identity Address<br>BLE_GAP_ADDR_RAND(0x01) Random Address or Random (Static) Identity Address |
| uint8_t | own_addr_type | Address type which local device uses in creating a link with the remote device.<br>macro / description<br>BLE_GAP_ADDR_PUBLIC(0x00) Public Address<br>BLE_GAP_ADDR_RAND(0x01) Random Address<br>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used.<br>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, the random address set by R_BLE_GAP_SetRandAddr(). |
| st_ble_gap_conn_phy_param_t * | p_conn_param_1M | Connection parameters for 1M PHY.<br><br>If this field is set to NULL, 1M PHY is not used in connecting. |
| st_ble_gap_conn_phy_param_t * | p_conn_param_2M | Connection parameters for 2M |

| | | |
|---|---|---|
| | | PHY.<br><br>If this field is set to NULL, 2M PHY is not used in connecting. |
| st_ble_gap_conn_phy_param_t * | p_conn_param_coded | Connection parameters for Coded PHY.<br><br>If this field is set to NULL, Coded PHY is not used in connecting. |

### ◆ st_ble_gap_rslv_list_key_set_t

| struct st_ble_gap_rslv_list_key_set_t | | |
|---|---|---|
| IRK of a remote device and IRK type of local device used in R_BLE_GAP_ConfRslvList(). | | |
| Data Fields | | |
| uint8_t | remote_irk[BLE_GAP_IRK_SIZE] | IRK of a remote device to be registered in the Resolving List. |
| uint8_t | local_irk_type | IRK type of the local device to be registered in the Resolving List.<br>macro: BLE_GAP_RL_LOC_KEY_ALL_ZERO(0x00) — All-zero IRK.<br>macro: BLE_GAP_RL_LOC_KEY_REGISTERED(0x01) — The IRK registered by R_BLE_GAP_SetLocIdInfo(). |

### ◆ st_ble_gap_set_phy_param_t

| struct st_ble_gap_set_phy_param_t | | |
|---|---|---|
| PHY configuration parameters used in R_BLE_GAP_SetPhy(). | | |
| Data Fields | | |
| uint8_t | tx_phys | Transmitter PHY preference.<br><br>The tx_phys field is set to a bitwise OR of the following values. All other values are ignored.<br>macro: BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01) — Use 1M PHY for Transmitter PHY. |

| | | |
|---|---|---|
| | | BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02) Use 2M PHY for Transmitter PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04) Use Coded PHY for Transmitter PHY. |
| uint8_t | rx_phys | Receiver PHY preference.<br>The rx_phys field is set to a bitwise OR of the following values. All other values are ignored.<br>macro / description<br>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01) Use 1M PHY for Receiver PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02) Use 2M PHY for Receiver PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04) Use Coded PHY for Receiver PHY. |
| uint16_t | phy_options | Coding scheme used in Coded PHY.<br>Select one of the following.<br>macro / description<br>BLE_GAP_SET_PHYS_OP_HOST_NO_PREF(0x00) No preferred coding.<br>BLE_GAP_SET_PHYS_OP_HOST_PREF_S_2(0x01) Use S=2 coding.<br>BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8(0x02) Use S=8 coding. |

◆ **st_ble_gap_set_def_phy_param_t**

struct st_ble_gap_set_def_phy_param_t

PHY preferences which allows a remote device to set used in R_BLE_GAP_SetDefPhy().

Data Fields

| | | |
|---|---|---|
| uint8_t | tx_phys | Transmitter PHY preferences which a remote device may change.<br><br>The tx_phys field is set to a bitwise OR of the following values. All other values are ignored.<br><br>macro: description<br>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01): Allow a remote device to set 1M PHY for transmitter PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02): Allow a remote device to set 2M PHY for transmitter PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04): Allow a remote device to set Coded PHY for transmitter PHY. |
| uint8_t | rx_phys | Receiver PHY preferences which a remote device may change.<br><br>The rx_phys field is set to a bitwise OR of the following values. All other values are ignored.<br><br>macro: description<br>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01): Allow a remote device to set 1M PHY for receiver PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x0: Allow a remote device to set 2M PHY |

| | | | |
|---|---|---|---|
| | | 2) | for receiver PHY. |
| | | BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04) | Allow a remote device to set Coded PHY for receiver PHY. |

### ◆ st_ble_gap_auth_info_t

struct st_ble_gap_auth_info_t

Pairing parameters required from a remote device or information about keys distributed from a remote device.

Data Fields

| | | |
|---|---|---|
| uint8_t | security | Security level.<br>value: 0x01 — The remote device requests Unauthenticated pairing.<br>value: 0x02 — The remote device requests Authenticated pairing. |
| uint8_t | pair_mode | Pairing mode.<br>value: 0x01 — The remote device requests Legacy pairing.<br>value: 0x02 — The remote device requests Secure Connections. |
| uint8_t | bonding | Bonding policy.<br>value: 0x00 — The remote device does not store the Bonding information.<br>value: 0x01 — The remote |

| | | |
|---|---|---|
| | | device stores the Bonding information. |
| uint8_t | ekey_size | Encryption key size. |

### ◆ st_ble_gap_key_dist_t

| struct st_ble_gap_key_dist_t | | |
|---|---|---|
| Keys distributed from a remote device. | | |
| Data Fields | | |
| uint8_t | enc_info[BLE_GAP_LTK_SIZE] | LTK. |
| uint8_t | mid_info[BLE_GAP_EDIV_SIZE +BLE_GAP_RAND_64_BIT_SIZE] | Ediv and rand. The first two bytes is ediv, the remaining bytes are rand. |
| uint8_t | id_info[BLE_GAP_IRK_SIZE] | IRK. |
| uint8_t | id_addr_info[ BLE_GAP_ID_ADDR_SIZE] | Identity address. The first byte is address type. The remaining bytes are device address. |
| uint8_t | sign_info[BLE_GAP_CSRK_SIZE] | CSRK. |

### ◆ st_ble_gap_key_ex_param_t

| struct st_ble_gap_key_ex_param_t | | |
|---|---|---|
| This structure includes the distributed keys and negotiated LTK size. | | |
| Data Fields | | |
| st_ble_gap_key_dist_t * | p_keys_info | Key information. |
| uint8_t | keys | Type of the distributed keys.<br><br>This field is a bitwise OR of the following values.<br><br>Bit Number / description<br>0 — LTK and Master Identification. LTK is distributed in Secure Connections, even if the bit is 1.<br>1 — IRK and Identity Address Information. |

| | | |
|---|---|---|
| | | 2 CSRK |
| uint8_t | ekey_size | The negotiated LTK size. |

### ◆ st_ble_gap_pairing_param_t

| struct st_ble_gap_pairing_param_t | | |
|---|---|---|
| Pairing parameters used in R_BLE_GAP_SetPairingParams(). | | |
| Data Fields | | |
| uint8_t | iocap | IO capabilities of local device.<br><br>Select one of the following.<br><br>macro / description<br>BLE_GAP_IOCAP_DISPLAY_ONLY(0x00) — Output function : Local device has the ability to display a 6 digit decimal number. Input function : None<br>BLE_GAP_IOCAP_DISPLAY_YESNO(0x01) — Output function : Output function : Local device has the ability to display a 6 digit decimal number. Input function : Local device has the ability to indicate 'yes' or 'no'<br>BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02) — Output function : None Input function : Local device has the ability to input the number '0' - '9'.<br>BLE_GAP_IOCAP_NOINPUT_ — Output function : |

| | | |
|---|---|---|
| | | NOOUTPUT(0x03) None Input function : None<br>BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04) Output function : Output function : Local device has the ability to display a 6 digit decimal number. Input function : Local device has the ability to input the number '0' - '9'. |
| uint8_t | mitm | MITM protection policy.<br>Select one of the following.<br>macro: description<br>BLE_GAP_SEC_MITM_BEST_EFFORT(0x00): MITM Protection not required.<br>BLE_GAP_SEC_MITM_STRICT (0x01): MITM Protection required. |
| uint8_t | bonding | Bonding policy.<br>macro: description<br>BLE_GAP_BONDING_NONE(0x00): Local device doesn't stores Bonding information.<br>BLE_GAP_BONDING (0x01): Local device stores Bonding information. |
| uint8_t | max_key_size | Maximum LTK size(in bytes).<br>Valid range is 7 - 16.<br>This field shall be set to a value not less than the min_key_size field. |
| uint8_t | min_key_size | Minimum LTK size(in bytes). |

<table>
<tr><td></td><td></td><td>Valid range is 7 - 16.<br>This field shall be set to a value not more than the max_key_size field.</td></tr>
<tr><td>uint8_t</td><td>loc_key_dist</td><td>Type of keys to be distributed from local device.<br><br>The loc_key_dist field is set to a bitwise OR of the following values.<br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_KEY_DIST_ENCKEY(0x01)</td><td>LTK</td></tr><tr><td>BLE_GAP_KEY_DIST_IDKEY(0x02)</td><td>IRK and Identity Address.</td></tr><tr><td>BLE_GAP_KEY_DIST_SIGNKEY(0x04)</td><td>CSRK</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>rem_key_dist</td><td>Type of keys which local device requests a remote device to distribute.<br><br>The rem_key_dist field is set to a bitwise OR of the following values.<br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_KEY_DIST_ENCKEY(0x01)</td><td>LTK. In case of Secure Connections, LTK is notified even if this bit is not set.</td></tr><tr><td>BLE_GAP_KEY_DIST_IDKEY(0x02)</td><td>IRK and Identity Address.</td></tr><tr><td>BLE_GAP_KEY_DIST_SIGNKEY(0x04)</td><td>CSRK</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>key_notf</td><td>Support for Key Press Notification in Passkey Entry.<br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SC_KEY_PRESS_N</td><td>Not support for Key Press</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| | | TF_NOT_SPRT(0x00) Notification.<br>BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01) Support for Key Press Notification. |
| uint8_t | sec_conn_only | Determine whether to accept only Secure Connections or not.<br>macro / description<br>BLE_GAP_SC_BEST_EFFORT(0x00) Accept Legacy pairing and Secure Connections.<br>BLE_GAP_SC_STRICT(0x01) Accept only Secure Connections. |

### ◆ st_ble_gap_oob_data_t

| struct st_ble_gap_oob_data_t | | |
|---|---|---|
| Oob data received from the remote device. This is used in R_BLE_GAP_SetRemOobData(). | | |
| Data Fields | | |
| uint8_t | legacy_oob[BLE_GAP_LEGACY_OOB_SIZE] | OOB data used in Legacy Pairing. |
| uint8_t | sc_cnf_val[BLE_GAP_OOB_CONFIRM_VAL_SIZE] | OOB confirmation value used in Secure Connections. |
| uint8_t | sc_rand[BLE_GAP_OOB_RANDOM_VAL_SIZE] | OOB rand used in Secure Connections. |

### ◆ st_ble_gap_cte_antenna_info_t

| struct st_ble_gap_cte_antenna_info_t |
|---|
| This is the parameters used in R_BLE_GAP_GetAntennaInfo(). |

### ◆ st_ble_gap_recv_test_param_t

| struct st_ble_gap_recv_test_param_t | | |
|---|---|---|
| This is the parameters used in R_BLE_GAP_ReceiverTest() | | |
| Data Fields | | |
| uint8_t | rx_ch | RF channel. |
| uint8_t | phy | The transmitter PHY of packets. |
| | | |

| | | |
|---|---|---|
| uint8_t | mod_idx | Whether or not the Controller should assume the receiver has a stable modulation index. |
| uint8_t | expect_cte_len | Expected length of the Constant Tone Extensions in received test reference packets. |
| uint8_t | expect_cte_type | Expected type of the Constant Tone Extensions in received test reference packets. |
| uint8_t | slot_duration | Switching and sampling slots durations. |
| uint8_t | switch_pattern_len | The number of Antenna IDs in the pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | Antenna ID in the pattern. |

### ◆ st_ble_gap_trans_test_param_t

| struct st_ble_gap_trans_test_param_t | | |
|---|---|---|
| This is the parameters used in R_BLE_GAP_TransmitterTest(). | | |
| Data Fields | | |
| uint8_t | tx_ch | RF channel. |
| uint8_t | test_data_len | Length of the Payload of the test reference packets. |
| uint8_t | packet_payload | Contents of the Payload of the test reference packets. |
| uint8_t | phy | The transmitter PHY of packets. |
| uint8_t | cte_len | Length of the Constant Tone Extension in the test reference packets. |
| uint8_t | cte_type | Type of the Constant Tone Extension in the test reference packets. |
| uint8_t | switch_pattern_len | The number of Antenna IDs in the pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | Antenna ID in the pattern. |
| int8_t | tx_power_level | Transmit power level to be used by the transmitter. |

### ◆ st_ble_gap_set_path_loss_rpt_param_t

| struct st_ble_gap_set_path_loss_rpt_param_t |
|---|
| This is the parameters used in R_BLE_GAP_SetPathLossReportingParam(). |

| Data Fields | | |
|---|---|---|
| uint16_t | conn_hdl | Connection handle. |
| uint8_t | high_thr | High threshold for the path loss. |
| uint8_t | high_hys | Hysteresis value for the high threshold. |
| uint8_t | low_thr | Low threshold for the path loss. |
| uint8_t | low_hys | Hysteresis value for the low threshold. |
| uint16_t | min_time_spent | Minimum time in number of connection events to be observed once the path loss crosses the threshold before an event is generated. |

### ◆ st_ble_gap_cte_connless_t

| struct st_ble_gap_cte_connless_t | | |
|---|---|---|
| connectionless CTE param | | |
| Data Fields | | |
| uint16_t | adv_hdl | For connectionless CTE handle is the Sync handle identifying the Periodic Sync that has been established. |
| uint8_t | cte_len | Length of CTE in 8us units.<br>Range: 0x02 to 0x14 |
| uint8_t | cte_type | CTE type.<br><br>Allowed values are:<br>BLE_GAP_CTE_TYPE_AOA<br>BLE_GAP_CTE_TYPE_AOD_1US<br>BLE_GAP_CTE_TYPE_AOD_2US |
| uint8_t | cte_count | Number of CTE to transmit in each periodic adv interval.<br>Range: 0x01 to 0x10 |
| uint8_t | pattern_len | Number of Antenna IDs in the switch pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | List of antenna IDs in the pattern. |

### ◆ st_ble_gap_cte_conn_t

| struct st_ble_gap_cte_conn_t | | |
|---|---|---|
| connection CTE param | | |
| Data Fields | | |
| | | |

| | | |
|---|---|---|
| uint16_t | conn_hdl | The ACL connection handle. |
| uint8_t | allow_cte_types | bitfield of CTE types that are allowed<br><br>bit should be one or compose of: BLE_GAP_CTE_TYPE_AOA (bit 0)<br>BLE_GAP_CTE_TYPE_AOD_1US (bit 1)<br>BLE_GAP_CTE_TYPE_AOD_2US (bit 2) |
| uint8_t | pattern_len | Number of Antenna IDs in the switch pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | List of antenna IDs in the pattern. |

### ◆ st_ble_gap_cte_connless_recv_t

| struct st_ble_gap_cte_connless_recv_t | | |
|---|---|---|
| connectionless CTE receive param | | |
| Data Fields | | |
| uint16_t | sync_hdl | Periodic Sync that has been established for CTE. |
| uint8_t | cte_type | CTE type.<br><br>CTE type should be get from st_ble_gap_perd_adv_rept_t of BLE_GAP_EVENT_ADV_REPT_IND event in connectionless CTE, or get from ACL connection in connection CTE. The slot_durations, pattern_len, and ant_ids parameters are only used when receiving an AoA Constant Tone Extension and do not affect the reception of an AoD Constant Tone Extension. |
| uint8_t | max_cte_count | Max number of CTEs to receive. Min is 1, max is 10, 0 means receive continuously. |
| uint8_t | slot_durations | Antenna switching slots. 1 for 1us or 2 for 2us |
| uint8_t | pattern_len | Length of antenna switch pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | Antenna switch pattern. |

### ◆ st_ble_gap_cte_conn_rx_param_t

| struct st_ble_gap_cte_conn_rx_param_t | | |
|---|---|---|
| connection CTE receive param | | |
| Data Fields | | |
| uint16_t | conn_hdl | For connection CTE handle is the ACL connection handle. |
| uint8_t | slot_durations | Antenna switching slots. 1 for 1us or 2 for 2us |
| uint8_t | pattern_len | Length of antenna switch pattern. |
| uint8_t | ant_ids[ BLE_GAP_CTE_MAX_ANTENNA] | Antenna switch pattern. |

### ◆ st_ble_gap_cte_conn_req_t

| struct st_ble_gap_cte_conn_req_t | | |
|---|---|---|
| connection CTE request | | |
| Data Fields | | |
| uint16_t | conn_hdl | For connection CTE handle is the ACL connection handle. |
| uint16_t | interval | Requested interval for initiating the CTE Request procedure.<br><br>Value 0x0 means, run the procedure once. Other values are intervals in number of connection events, to run the command periodically. |
| uint8_t | cte_length | Requested length of the CTE in 8 us units. |
| uint8_t | cte_type | Requested type of the CTE.<br><br>Allowed values are BLE_GAP_CTE_TYPE_AOA, BLE_GAP_CTE_TYPE_AOD_1US and BLE_GAP_CTE_TYPE_AOD_2US |

### ◆ st_ble_gap_subrate_param_t

| struct st_ble_gap_subrate_param_t |
|---|
| subrating param |

### ◆ st_ble_gap_ver_num_t

| struct st_ble_gap_ver_num_t |
|---|
| Version number of host stack. |

| Data Fields | | |
|---|---|---|
| uint8_t | major | Major version number. |
| uint8_t | minor | Minor version number. |
| uint8_t | subminor | Subminor version number. |

### ◆ st_ble_gap_loc_ver_info_t

| struct st_ble_gap_loc_ver_info_t | | |
|---|---|---|
| Version number of Controller.<br>Refer Bluetooth SIG Assigned Number(https://www.bluetooth.com/specifications/assigned-numbers). | | |
| Data Fields | | |
| uint8_t | hci_ver | Bluetooth HCI version. |
| uint16_t | hci_rev | Bluetooth HCI revision. |
| uint8_t | lmp_ver | Link Layer revision. |
| uint16_t | mnf_name | Manufacturer ID. |
| uint16_t | lmp_sub_ver | Link Layer subversion. |

### ◆ st_ble_gap_loc_dev_info_evt_t

| struct st_ble_gap_loc_dev_info_evt_t | | |
|---|---|---|
| Version information of local device. | | |
| Data Fields | | |
| st_ble_dev_addr_t | l_dev_addr | Bluetooth Device Address. |
| st_ble_gap_ver_num_t | l_ver_num | Version number of host stack in local device. |
| st_ble_gap_loc_ver_info_t | l_bt_info | Version number of Controller in local device. |

### ◆ st_ble_gap_hw_err_evt_t

| struct st_ble_gap_hw_err_evt_t | | |
|---|---|---|
| Hardware error that is notified from Controller. | | |
| Data Fields | | |
| uint8_t | hw_code | The hw_code field indicates the cause of the hardware error. |

### ◆ st_ble_gap_cmd_err_evt_t

| struct st_ble_gap_cmd_err_evt_t | | |
|---|---|---|
| HCI Command error. | | |
| Data Fields | | |

| | | |
|---|---|---|
| uint16_t | op_code | The opcode of HCI Command which caused the error. |
| uint32_t | module_id | Module ID which caused the error. |

### ◆ st_ble_gap_adv_rept_t

| struct st_ble_gap_adv_rept_t | | |
|---|---|---|
| Advertising Report. | | |
| Data Fields | | |
| uint8_t | num | The number of Advertising Reports received. |
| uint8_t | adv_type | Type of Advertising Packet.<br>valuer: 0x00 — Connectable and scannable undirected advertising(ADV_IND).<br>0x01 — Connectable directed advertising(ADV_DIRECT_IND).<br>0x02 — Scannable undirected advertising(ADV_SCAN_IND).<br>0x03 — Non-connectable undirected advertising(ADV_NONCONN_IND).<br>0x04 — Scan response(SCAN_RSP). |
| uint8_t | addr_type | Address type of the advertiser.<br>value: 0x00 — Public Address.<br>0x01 — Random Address.<br>0x02 — Public Identity Address which could be resolved in |

| | | |
|---|---|---|
| | | Controller.<br>0x03 Random Identity Address which could be resolved in Controller. |
| uint8_t * | p_addr | Address of the advertiser.<br>*Note*<br>*The BD address setting format is little endian.* |
| uint8_t | len | Length of Advertising data(in bytes).<br>Valid range is 0 - 31. |
| int8_t | rssi | RSSI(in dBm).<br>Valid range is -127 <= tx_pwr <= 20 and 127.<br>If the tx_pwr is 127, it means that RSSI could not be retrieved. |
| uint8_t * | p_data | Advertising data/Scan Response Data. |

### ◆ st_ble_gap_ext_adv_rept_t

| struct st_ble_gap_ext_adv_rept_t | | |
|---|---|---|
| Extended Advertising Report. | | |
| Data Fields | | |
| uint8_t | num | The number of Advertising Reports received. |
| uint16_t | adv_type | Type of Advertising Packet.<br>Bit Number: description<br>0: Connectable advertising.<br>1: Scannable advertising.<br>2: Directed advertising.<br>3: Scan response.<br>4: Legacy advertising PDU. |

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>5-6</td><td>The status of Advertising Data/Scan Response Data.<br>Data Status:<br>00b = Complete<br>01b = Incomplete, more data come<br>10b = Incomplete, data truncated, no more to come</td></tr>
<tr><td>All other bits</td><td>Reserved for future use</td></tr>
</table>
</td></tr>
<tr><td>uint8_t</td><td>addr_type</td><td>Address type of the advertiser.
<table>
<tr><th>value</th><th>description</th></tr>
<tr><td>0x00</td><td>Public Address.</td></tr>
<tr><td>0x01</td><td>Random Address.</td></tr>
<tr><td>0x02</td><td>Public Identity Address which could be resolved in Controller.</td></tr>
<tr><td>0x03</td><td>Random Identity Address which could be resolved in Controller.</td></tr>
<tr><td>0xFF</td><td>Anonymous advertisement.</td></tr>
</table>
</td></tr>
<tr><td>uint8_t *</td><td>p_addr</td><td>Address of the advertiser.<br><br>Note<br>The BD address setting format is little endian.</td></tr>
<tr><td>uint8_t</td><td>adv_phy</td><td>The primary PHY configuration of the advertiser.</td></tr>
</table>

| | | |
|---|---|---|
| | | The primary PHY configuration of the advertiser.<br>value: 0x01 — 1M PHY<br>value: 0x03 — Coded PHY |
| uint8_t | sec_adv_phy | The secondary PHY configuration of the advertiser.<br>value: 0x00 — Nothing has been received with Secondary Advertising Channel.<br>value: 0x01 — The Secondary Advertising PHY configuration was 1M PHY.<br>value: 0x02 — The Secondary Advertising PHY configuration was 2M PHY.<br>value: 0x03 — The Secondary Advertising PHY configuration was Coded PHY. |
| uint8_t | adv_sid | Advertising SID included in the received Advertising Report.<br><br>Valid range is 0 <= adv_sid <= 0x0F and 0xFF.<br>If the adv_sid is 0xFF, there is no field which includes SID. |
| int8_t | tx_pwr | TX power(in dBm).<br><br>Valid range is -127 <= tx_pwr <= 20 and 127.<br>If the tx_pwr is 127, it means that Tx power could not be retrieved. |

<table>
<tr><td>int8_t</td><td>rssi</td><td>RSSI(in dBm).<br><br>Valid range is -127 <= tx_pwr <= 20 and 127.<br>If the tx_pwr is 127, it means that RSSI could not be retrieved.</td></tr>
<tr><td>uint16_t</td><td>perd_adv_intv</td><td>Periodic Advertising interval.<br><br>If the perd_adv_intv is 0x0000, it means that this advertising is not periodic advertising.<br>If the perd_adv_intv is 0x0006 - 0xFFFF, it means that this field is the Periodic Advertising interval.<br>Periodic Advertising interval = per_adv_intr * 1.25ms.</td></tr>
<tr><td>uint8_t</td><td>dir_addr_type</td><td>The address type of Direct Advertising PDU.<table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>Public Address.</td></tr><tr><td>0x01</td><td>Random Address.</td></tr><tr><td>0x02</td><td>Public Identity Address which could be resolved in Controller.</td></tr><tr><td>0x03</td><td>Random Identity Address which could be resolved in Controller.</td></tr><tr><td>0xFE</td><td>Resolvable Privacy Address which could not be resolved in Controller.</td></tr></table></td></tr>
<tr><td>uint8_t *</td><td>p_dir_addr</td><td>Address of Direct Advertising PDU.<br><br>Note<br>The BD address setting format is little endian.</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

| | | |
|---|---|---|
| uint8_t | len | Length of Advertising data(in bytes).<br><br>Valid range is 0 - 229. |
| uint8_t * | p_data | Advertising data/Scan Response Data. |

### ◆ st_ble_gap_perd_adv_rept_t

| struct st_ble_gap_perd_adv_rept_t | | |
|---|---|---|
| Periodic Advertising Report. | | |
| Data Fields | | |
| uint16_t | sync_hdl | Sync handle.<br><br>Valid range is 0x0000 - 0x0EFF. |
| int8_t | tx_pwr | TX power(in dBm).<br><br>Valid range is -127 <= tx_pwr <= 20 and 127.<br>If tx_pwr is 127, it means that Tx power could not be retrieved. |
| int8_t | rssi | RSSI(in dBm).<br><br>Valid range is -127 <= rssi <= 20 and 127.<br>If rssi is 127, it means that RSSI could not be retrieved. |
| uint8_t | cte_type | Type of Constant Tone Extension in the periodic advertising packets. |

| value | description |
|---|---|
| 0x00 | AoA Constant Tone Extension. |
| 0x01 | AoD Constant Tone Extension with 1 µs slots. |
| 0x02 | AoD Constant Tone Extension with 2 µs slots. |
| 0xFF | No Constant Tone Extension. |

| | | |
|---|---|---|
| uint8_t | data_status | Reserved for future use.<br>value: 0x00 — Data Complete.<br>0x01 — Data incomplete, more data to come.<br>0x02 — Data incomplete, data truncated, no more to come. |
| uint8_t | len | Length of Periodic Advertising data(in bytes).<br>Valid range is 0 - 247. |
| uint8_t * | p_data | Periodic Advertising data. |

### ◆ st_ble_gap_adv_rept_evt_t

| struct st_ble_gap_adv_rept_evt_t | | |
|---|---|---|
| Advertising report. | | |
| Data Fields | | |
| uint8_t | adv_rpt_type | Data type.<br>value: 0x00 — Advertising Report.<br>0x01 — Extended Advertising Report.<br>0x02 — Periodic Advertising Report.<br>If the BLE Protocol Stack library type is "extended", the adv_rpt_type field in a Legacy Advertising Report event is 0x01. |
| union st_ble_gap_adv_rept_evt_t | param | Advertising Report. |

### ◆ st_ble_gap_adv_rept_evt_t.param

| union st_ble_gap_adv_rept_evt_t.param | | |
|---|---|---|
| Advertising Report. | | |
| Data Fields | | |
| st_ble_gap_adv_rept_t * | p_adv_rpt | Advertising Report. |
| st_ble_gap_ext_adv_rept_t * | p_ext_adv_rpt | Extended Advertising Report. |
| st_ble_gap_perd_adv_rept_t * | p_per_adv_rpt | Periodic Advertising Report. |

### ◆ st_ble_gap_adv_set_evt_t

| struct st_ble_gap_adv_set_evt_t | | |
|---|---|---|
| Advertising handle. | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle specifying the advertising set configured advertising parameters. |

### ◆ st_ble_gap_adv_off_evt_t

| struct st_ble_gap_adv_off_evt_t | | |
|---|---|---|
| Information about the advertising set which stops advertising. | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set which has stopped advertising.<br><br>Valid range is 0x00 - 0x03. |
| uint8_t | reason | The reason for stopping advertising.<br>value: 0x01 — description: Advertising has been stopped by R_BLE_GAP_StopAdv().<br>value: 0x02 — description: Because the duration specified by R_BLE_GAP_StartAdv() was expired, advertising has terminated.<br>value: 0x03 — description: Because the max_extd_adv_evts |

| | | |
|---|---|---|
| | | parameter specified by R_BLE_GAP_StartAdv() was reached, advertising has terminated.<br>0x04 Because the connection was established with the remote device, advertising has terminated. |
| uint16_t | conn_hdl | Connection handle.<br><br>If the reason field is 0x04, this field indicates connection handle identifying the remote device connected with local device. If other reasons, ignore this field. |
| uint8_t | num_comp_ext_adv_evts | The number of the advertising event that has been received until advertising has terminated.<br><br>If max_extd_adv_evts by R_BLE_GAP_StartAdv() is not 0, this parameter is valid. |

◆ **st_ble_gap_adv_data_evt_t**

| struct st_ble_gap_adv_data_evt_t | | |
|---|---|---|
| This structure notifies that advertising data has been set to Controller by R_BLE_GAP_SetAdvSresData(). | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set to which advertising data/scan response data/periodic advertising data is set. |
| uint8_t | data_type | Type of the data set to the advertising set.<br>value: description<br>BLE_GAP_ADV: Advertising |

| | | |
|---|---|---|
| | | _DATA_MODE(0x00) data<br>BLE_GAP_SCAN_RSP_DATA_MODE(0x01) Scan response data<br>BLE_GAP_PERD_ADV_DATA_MODE(0x02) Periodic advertising data |

### ◆ st_ble_gap_rem_adv_set_evt_t

| struct st_ble_gap_rem_adv_set_evt_t | | |
|---|---|---|
| This structure notifies that an advertising set has been removed. | | |
| Data Fields | | |
| uint8_t | remove_op | This field indicates that the advertising set has been removed or cleared.<br>value: description<br>0x01: The advertising set has been removed.<br>0x02: The advertising set has been cleared. |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set which has been removed.<br><br>If the advertising set has been cleared, this field is ignored. |

### ◆ st_ble_gap_conn_evt_t

| struct st_ble_gap_conn_evt_t | | |
|---|---|---|
| This structure notifies that a link has been established. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the created link. |
| uint8_t | role | The role of the link.<br>value: description<br>0x00: Master<br>0x01: Slave |
| | | |

<table>
<tr>
<td>uint8_t</td>
<td>remote_addr_type</td>
<td>Address type of the remote device.
<table>
<tr><th>value</th><th>description</th></tr>
<tr><td>0x00</td><td>Public Address</td></tr>
<tr><td>0x01</td><td>Random Address</td></tr>
<tr><td>0x02</td><td>Public Identity Address.<br>It indicates that the Controller could resolve the resolvable private address of the remote device.</td></tr>
<tr><td>0x03</td><td>Random Identity Address.<br>It indicates that the Controller could resolve the resolvable private address of the remote device.</td></tr>
</table>
</td>
</tr>
<tr>
<td>uint8_t</td>
<td>remote_addr[<br>BLE_BD_ADDR_LEN]</td>
<td>Address of the remote device.<br><br>Note<br>The BD address setting format is little endian.</td>
</tr>
<tr>
<td>uint8_t</td>
<td>local_rpa[BLE_BD_ADDR_LEN]</td>
<td>Resolvable private address that local device used in connection procedure.<br><br>The local device address used in creating the link when the address type was set to BLE_GAP_ADDR_RPA_ID_PUBLIC or BLE_GAP_ADDR_RPA_ID_RANDOM by R_BLE_GAP_SetAdvParam() or R_BLE_GAP_CreateConn(). If the address type was set to other than BLE_GAP_ADDR_RPA_ID_PUBLIC and BLE_GAP_ADDR_RPA_ID_RA</td>
</tr>
</table>

| | | |
|---|---|---|
| | | NDOM, this field is set to all-zero.<br><br>*Note*<br>*The BD address setting format is little endian.* |
| uint8_t | remote_rpa[BLE_BD_ADDR_LEN] | Resolvable private address that the remote device used in connection procedure.<br><br>This field indicates the remote resolvable private address when remote_addr_type is 0x02 or 0x03. If remote_addr_type is other than 0x02 and 0x03, this field is set to all-zero.<br><br>*Note*<br>*The BD address setting format is little endian.* |
| uint16_t | conn_intv | Connection interval.<br><br>Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv * 1.25. |
| uint16_t | conn_latency | Slave latency.<br><br>Valid range is 0x0000 - 0x01F3. |
| uint16_t | sup_to | Supervision timeout.<br><br>Valid range is 0x000A - 0x0C80.Time(ms) = sup_to * 10. |
| uint8_t | clk_acc | Master_Clock_Accuracy.<br>value / description<br>0x00 500ppm<br>0x01 250ppm<br>0x02 150ppm<br>0x03 100ppm<br>0x04 75ppm<br>0x05 50ppm<br>0x06 30ppm<br>0x07 20ppm |

### ◆ st_ble_gap_disconn_evt_t

| struct st_ble_gap_disconn_evt_t | | |
|---|---|---|
| This structure notifies that a link has been disconnected. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link disconnected. |
| uint8_t | reason | The reason for disconnection.<br><br>Refer Core Specification Vol.2 Part D ,"2 Error Code Descriptions". |

◆ **st_ble_gap_rd_ch_map_evt_t**

| struct st_ble_gap_rd_ch_map_evt_t | | |
|---|---|---|
| This structure notifies that Channel Map has been retrieved by R_BLE_GAP_ReadChMap(). | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link whose Channel Map was retrieved. |
| uint8_t | ch_map[BLE_GAP_CH_MAP_SIZE ] | Channel Map. |

◆ **st_ble_gap_rd_rssi_evt_t**

| struct st_ble_gap_rd_rssi_evt_t | | |
|---|---|---|
| This structure notifies that RSSI has been retrieved by R_BLE_GAP_ReadRssi(). | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link whose RSSI was retrieved. |
| int8_t | rssi | RSSI(in dBm).<br><br>Valid range is -127 < rssi < 20 and 127.<br>If this field is 127, it indicates that RSSI could not be retrieved. |

◆ **st_ble_gap_dev_info_evt_t**

| struct st_ble_gap_dev_info_evt_t | | |
|---|---|---|
| This structure notifies that information about remote device has been retrieved by R_BLE_GAP_GetRemDevInfo(). | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device whose |

| | | |
|---|---|---|
| | | information has been retrieved. |
| uint8_t | get_status | Information about the remote device. This field is a bitwise OR of the following values.<br>Bit Number: description<br>bit0: Address<br>bit1: Version, company_id, subversion<br>bit2: Feature<br>All other bits: Reserved for future use |
| st_ble_dev_addr_t | addr | Address of the remote device. |
| uint8_t | version | The version of Link Layer of the remote device.<br><br>Refer to Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers) regarding defined number. |
| uint16_t | company_id | The manufacturer ID of the remote device.<br><br>Refer to Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers) regarding defined number. |
| uint16_t | subversion | The subversion of Link Layer. |
| uint8_t | features[ BLE_GAP_REM_FEATURE_SIZE] | LE feature supported in the remote device.<br><br>Refer to Core Spec Vol 6, Part B 4.6 FEATURE SUPPORT. |

### ◆ st_ble_gap_conn_upd_evt_t

| struct st_ble_gap_conn_upd_evt_t | | |
|---|---|---|
| This structure notifies that connection parameters has been updated. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the connection whose parameters has been updated. |

| | | |
|---|---|---|
| uint16_t | conn_intv | Updated Connection Interval.<br><br>Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv * 1.25. |
| uint16_t | conn_latency | Updated slave latency.<br><br>Valid range is 0x0000 - 0x01F3. |
| uint16_t | sup_to | Updated supervision timeout.<br><br>Valid range is 0x000A - 0x0C80. Time(ms) = sup_to * 10. |

### ◆ st_ble_gap_conn_upd_req_evt_t

| struct st_ble_gap_conn_upd_req_evt_t | | |
|---|---|---|
| This structure notifies that a request for connection parameters update has been received. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link that was requested to update connection parameters. |
| uint16_t | conn_intv_min | Minimum connection interval.<br><br>Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv_min * 1.25. |
| uint16_t | conn_intv_max | Maximum connection interval.<br><br>Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv_max * 1.25. |
| uint16_t | conn_latency | Slave latency.<br><br>Valid range is 0x0000 - 0x01F3. |
| uint16_t | sup_to | Supervision timeout.<br><br>Valid range is 0x000A - 0x0C80. Time(ms) = sup_to * 10 |

### ◆ st_ble_gap_conn_hdl_evt_t

| struct st_ble_gap_conn_hdl_evt_t | | |
|---|---|---|
| This structure notifies that a GAP Event that includes only connection handle has occurred. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle. |

### ◆ st_ble_gap_data_len_chg_evt_t

| struct st_ble_gap_data_len_chg_evt_t | | |
|---|---|---|
| This structure notifies that the packet data length has been updated. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link that updated Data Length. |
| uint16_t | tx_octets | Updated transmission packet size(in bytes).<br><br>Valid range is 0x001B - 0x00FB. |
| uint16_t | tx_time | Updated transmission time(us).<br><br>Valid range is 0x0148 - 0x4290. |
| uint16_t | rx_octets | Updated receive packet size(in bytes).<br><br>Valid range is 0x001B - 0x00FB. |
| uint16_t | rx_time | Updated receive time(us).<br><br>Valid range is 0x0148 - 0x4290. |

### ◆ st_ble_gap_rd_rpa_evt_t

| struct st_ble_gap_rd_rpa_evt_t | | |
|---|---|---|
| This structure notifies that the local resolvable private address has been retrieved by R_BLE_GAP_ReadRpa(). | | |
| Data Fields | | |
| st_ble_dev_addr_t | addr | The resolvable private address of local device. |

### ◆ st_ble_gap_phy_upd_evt_t

| struct st_ble_gap_phy_upd_evt_t | | |
|---|---|---|
| This structure notifies that PHY for a connection has been updated. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link that has been updated. |
| uint8_t | tx_phy | Transmitter PHY.<br>value: 0x01 — description: The transmitter |

| | | |
|---|---|---|
| | | PHY has been updated to 1M PHY.<br>0x02 The transmitter PHY has been updated to 2M PHY.<br>0x03 The transmitter PHY has been updated to Coded PHY. |
| uint8_t | rx_phy | Receiver PHY.<br>value / description<br>0x01 The receiver PHY has been updated to 1M PHY.<br>0x02 The receiver PHY has been updated to 2M PHY.<br>0x03 The receiver PHY has been updated to Coded PHY. |

### ◆ st_ble_gap_phy_rd_evt_t

| struct st_ble_gap_phy_rd_evt_t | | |
|---|---|---|
| This structure notifies that the PHY settings has been retrieved by R_BLE_GAP_ReadPhy(). | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link that has been retrieved the PHY settings. |
| uint8_t | tx_phy | Transmitter PHY.<br>value / description<br>0x01 The transmitter PHY has been updated to 1M PHY.<br>0x02 The transmitter PHY has been |

| | | |
|---|---|---|
| | | updated to 2M PHY.<br>0x03 The transmitter PHY has been updated to Coded PHY. |
| uint8_t | rx_phy | Receiver PHY.<br>value: description<br>0x01 The receiver PHY has been updated to 1M PHY.<br>0x02 The receiver PHY has been updated to 2M PHY.<br>0x03 The receiver PHY has been updated to Coded PHY. |

### ◆ st_ble_gap_scan_req_recv_evt_t

| struct st_ble_gap_scan_req_recv_evt_t | | |
|---|---|---|
| This structure notifies that a Scan Request packet has been received from a Scanner. | | |
| Data Fields | | |
| uint8_t | adv_hdl | Advertising handle identifying the advertising set that has received the Scan Request. |
| uint8_t | scanner_addr_type | Address type of the Scanner.<br>value: description<br>0x00 Public Address.<br>0x01 Random Address.<br>0x02 Public Identity Address which could be resolved in Controller.<br>0x03 Random Identity Address which could be resolved in |

| | | |
|---|---|---|
| | | Controller. |
| uint8_t | scanner_addr[ BLE_BD_ADDR_LEN] | Address of the Scanner.<br>*Note* *The BD address setting format is little endian.* |

### ◆ st_ble_gap_sync_est_evt_t

| struct st_ble_gap_sync_est_evt_t | | |
|---|---|---|
| This structure notifies that a Periodic sync has been established. | | |
| Data Fields | | |
| uint16_t | sync_hdl | Sync handle identifying the Periodic Sync that has been established. |
| uint8_t | adv_sid | Advertising SID identifying the advertising set that has established the Periodic Sync. |
| uint8_t | adv_addr_type | Address type of the advertiser.<br>value: description<br>0x00: Public Address.<br>0x01: Random Address.<br>0x02: Public Identity Address which could be resolved in Controller.<br>0x03: Random Identity Address which could be resolved in Controller. |
| uint8_t * | p_adv_addr | Address of the advertiser.<br>*Note* *The BD address setting format is little endian.* |
| uint8_t | adv_phy | Advertising PHY.<br>value: description<br>0x01: Advertiser PHY is 1M PHY. |

| | | |
|---|---|---|
| | | 0x02 Advertiser PHY is 2M PHY.<br>0x03 Advertiser PHY is Coded PHY. |
| uint16_t | perd_adv_intv | Periodic Advertising Interval.<br>Valid range is 0x0006 - 0xFFFF. Time(ms) = perd_adv_intv * 1.25. |
| uint8_t | adv_clk_acc | Advertiser Clock Accuracy.<br>value / description<br>0x00 500ppm<br>0x01 250ppm<br>0x02 150ppm<br>0x03 100ppm<br>0x04 75ppm<br>0x05 50ppm<br>0x06 30ppm<br>0x07 20ppm |

### ◆ st_ble_gap_sync_hdl_evt_t

| struct st_ble_gap_sync_hdl_evt_t | | |
|---|---|---|
| This structure notifies that a GAP Event that includes only sync handle has occurred. | | |
| Data Fields | | |
| uint16_t | sync_hdl | Sync handle. |

### ◆ st_ble_gap_white_list_conf_evt_t

| struct st_ble_gap_white_list_conf_evt_t | | |
|---|---|---|
| This structure notifies that White List has been configured. | | |
| Data Fields | | |
| uint8_t | op_code | The operation for White List.<br>value / description<br>0x01 A device was added to White List.<br>0x02 A device was deleted from White List. |

| | | |
|---|---|---|
| | | 0x03 White List was cleared. |
| uint8_t | num | The number or devices which have been added to or deleted from White List. |

### ◆ st_ble_gap_rslv_list_conf_evt_t

| struct st_ble_gap_rslv_list_conf_evt_t | | |
|---|---|---|
| This structure notifies that Resolving List has been configured. | | |
| Data Fields | | |
| uint8_t | op_code | The operation for Resolving List.<br>value: description<br>0x01: A device was added to Resolving List.<br>0x02: A device was deleted from Resolving List.<br>0x03: Resolving List was cleared. |
| uint8_t | num | The number or devices which have been added to or deleted from Resolving List. |

### ◆ st_ble_gap_perd_list_conf_evt_t

| struct st_ble_gap_perd_list_conf_evt_t | | |
|---|---|---|
| This structure notifies that Periodic Advertiser List has been configured. | | |
| Data Fields | | |
| uint8_t | op_code | The operation for Periodic Advertiser List.<br>value: description<br>0x01: A device was added to Periodic Advertiser List.<br>0x02: A device was deleted from Periodic Advertiser List.<br>0x03: Periodic |

| | | |
|---|---|---|
| | | Advertiser List was cleared. |
| uint8_t | num | The number or devices which have been added to or deleted from Periodic Advertiser List. |

### ◆ st_ble_gap_set_priv_mode_evt_t

| struct st_ble_gap_set_priv_mode_evt_t | | |
|---|---|---|
| This structure notifies that Privacy Mode has been configured. | | |
| Data Fields | | |
| uint8_t | num | The number or devices which have been set privacy mode. |

### ◆ st_ble_gap_pairing_req_evt_t

| struct st_ble_gap_pairing_req_evt_t | | |
|---|---|---|
| This structure notifies that a pairing request from a remote device has been received. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device that sent the pairing request. |
| st_ble_dev_addr_t | bd_addr | The address of the remote device. |
| st_ble_gap_auth_info_t | auth_info | The Pairing parameters of the remote device. |

### ◆ st_ble_gap_passkey_display_evt_t

| struct st_ble_gap_passkey_display_evt_t | | |
|---|---|---|
| This structure notifies that a request for Passkey display in pairing has been received. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device that requested Passkey display. |
| uint32_t | passkey | Passkey.<br><br>This field is a 6 digit decimal number(000000-999999). |

### ◆ st_ble_gap_num_comp_evt_t

| struct st_ble_gap_num_comp_evt_t | | |
|---|---|---|
| This structure notifies that a request for Numeric Comparison in pairing has been received. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying |

| | | |
|---|---|---|
| | | the remote device that requested Numeric Comparison. |
| uint32_t | numeric | The number to be confirmed in Numeric Comparison.<br><br>This field is a 6 digit decimal number(000000-999999). |

### ◆ st_ble_gap_key_press_ntf_evt_t

| struct st_ble_gap_key_press_ntf_evt_t | | |
|---|---|---|
| This structure notifies that the remote device has input a key in Passkey Entry. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device that input a key. |
| uint8_t | key_type | Type of the key that the remote device input.<br>value: description<br>0x00: Passkey entry started.<br>0x01: Passkey digit entered.<br>0x02: Passkey digit erased.<br>0x03: Passkey cleared.<br>0x04: Passkey entry completed. |

### ◆ st_ble_gap_pairing_info_evt_t

| struct st_ble_gap_pairing_info_evt_t | | |
|---|---|---|
| This structure notifies that the pairing has completed. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device that the pairing has been done with. |
| st_ble_dev_addr_t | bd_addr | Address of the remote device. |
| st_ble_gap_auth_info_t | auth_info | Key information exchanged in pairing.<br><br>If local device supports bonding, store the information |

| | | |
|---|---|---|
| | | in non-volatile memory in order to set it to host stack after power re-supply. |

### ◆ st_ble_gap_enc_chg_evt_t

| struct st_ble_gap_enc_chg_evt_t | | |
|---|---|---|
| This structure notifies that the encryption status of a link has been changed. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the link that has been changed. |
| uint8_t | enc_status | Encryption Status.<br>value: 0x00 — description: Encryption OFF.<br>value: 0x01 — description: Encryption ON.<br>value: 0x03 — description: Encryption updated by Encryption Key Refresh Completed. |

### ◆ st_ble_gap_peer_key_info_evt_t

| struct st_ble_gap_peer_key_info_evt_t | | |
|---|---|---|
| This structure notifies that the remote device has distributed the keys. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device that has distributed the keys. |
| st_ble_dev_addr_t | bd_addr | Address of the remote device. |
| st_ble_gap_key_ex_param_t | key_ex_param | Distributed keys.<br><br>If local device supports bonding, store the keys in non-volatile memory and at power re-supply set to the host stack by R_BLE_GAP_SetBondInfo(). |

### ◆ st_ble_gap_ltk_req_evt_t

| struct st_ble_gap_ltk_req_evt_t | | |
|---|---|---|
| This structure notifies that a LTK request from a remote device has been received. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying |

| | | |
|---|---|---|
| | | the remote device which requests for the LTK. |
| uint16_t | ediv | Ediv. |
| uint8_t * | p_peer_rand | Rand. |

### ◆ st_ble_gap_ltk_rsp_evt_t

| struct st_ble_gap_ltk_rsp_evt_t | | |
|---|---|---|
| This structure notifies that local device has replied to the LTK request from the remote device. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device to be sent the response to the LTK request. |
| uint8_t | response | The response to the LTK request.<br>value: description<br>0x00: Local device replied with the stored LTK.<br>0x01: Local device rejected the LTK request, because the LTK was not found. |

### ◆ st_ble_gap_sc_oob_data_evt_t

| struct st_ble_gap_sc_oob_data_evt_t | | |
|---|---|---|
| This structure notifies that OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData(). | | |
| Data Fields | | |
| uint8_t * | p_sc_oob_conf | Confirmation value(16 bytes) of OOB Data. |
| uint8_t * | p_sc_oob_rand | Rand(16bytes) of OOB Data. |

### ◆ st_ble_gap_bond_info_t

| struct st_ble_gap_bond_info_t | | |
|---|---|---|
| Bonding information used in R_BLE_GAP_SetBondInfo(). | | |
| Data Fields | | |
| st_ble_dev_addr_t * | p_addr | Address of the device which exchanged the keys. |

| | | |
|---|---|---|
| st_ble_gap_auth_info_t * | p_auth_info | Information about the keys. |
| st_ble_gap_key_ex_param_t * | p_keys | Keys distributed from the remote device in paring. |

### ◆ st_cte_iq_sample_t

| struct st_cte_iq_sample_t |
|---|
| CTE IQ sample data. |

### ◆ st_ble_gap_cte_connless_rept_t

| struct st_ble_gap_cte_connless_rept_t |
|---|
| connectionless CTE data report |

### ◆ st_ble_gap_cte_conn_rept_t

| struct st_ble_gap_cte_conn_rept_t |
|---|
| connection CTE data report |

### ◆ st_ble_subrate_upd_t

| struct st_ble_subrate_upd_t |
|---|
| subrating update event |

### ◆ st_ble_gap_past_est_evt_t

| struct st_ble_gap_past_est_evt_t | | |
|---|---|---|
| This structure notifies that. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Periodic Advertising Sync Transfer options |
| uint16_t | service_data | |
| st_ble_gap_sync_est_evt_t | sync | |

### ◆ st_ble_gap_tx_power_reporting_evt_t

| struct st_ble_gap_tx_power_reporting_evt_t |
|---|
| This structure notifies that. |

### ◆ st_ble_gap_pass_loss_thr_evt_t

| struct st_ble_gap_pass_loss_thr_evt_t |
|---|
| This structure notifies that a path loss report has been received. |

### ◆ st_ble_gap_req_peer_sca_evt_t

| struct st_ble_gap_req_peer_sca_evt_t |
|---|
| This structure notifies that a SCA request to a remote device has been completed. |

### ◆ st_ble_gap_dtm_test_end_evt_t

| struct st_ble_gap_dtm_test_end_evt_t | | |
|---|---|---|
| report of dtm transmit/receive test end | | |
| Data Fields | | |
| uint16_t | recv_cnt | the number of received packets |

### ◆ st_ble_gap_enhanced_read_tx_power_level_evt_t

| struct st_ble_gap_enhanced_read_tx_power_level_evt_t | | |
|---|---|---|
| Power level report of remove device. | | |
| Data Fields | | |
| uint16_t | handle | Indicate an ACL connection. |
| uint8_t | phy | PHY. |
| int8_t | crt_tx_power_level | Current transmit power level. |
| int8_t | max_tx_power_level | Maximum transmit power level. |

## Macro Definition Documentation

### ◆ BLE_BD_ADDR_LEN

| #define BLE_BD_ADDR_LEN |
|---|
| Bluetooth Device Address Size |

### ◆ BLE_MASTER

| #define BLE_MASTER |
|---|
| Master Role. |

### ◆ BLE_SLAVE

| #define BLE_SLAVE |
|---|
| Slave Role. |

### ◆ BLE_GAP_ADDR_PUBLIC

| #define BLE_GAP_ADDR_PUBLIC |
| --- |
| Public Address. |

### ◆ BLE_GAP_ADDR_RAND

| #define BLE_GAP_ADDR_RAND |
| --- |
| Random Address. |

### ◆ BLE_GAP_ADDR_RPA_ID_PUBLIC

| #define BLE_GAP_ADDR_RPA_ID_PUBLIC |
| --- |
| Resolvable Private Address.<br>If the IRK of local device has not been registered in Resolving List, public address is used. |

### ◆ BLE_GAP_ADDR_RPA_ID_RANDOM

| #define BLE_GAP_ADDR_RPA_ID_RANDOM |
| --- |
| Resolvable Private Address.<br>If the IRK of local device has not been registered in Resolving List, random address is used. |

### ◆ BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST

| #define BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST |
| --- |
| Accept all advertising and scan response PDUs.<br>The following are excluded:<br>Advertising and scan response PDUs where the advertiser's identity address is not in the White List.<br>Directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted. |

### ◆ BLE_GAP_IOCAP_DISPLAY_ONLY

| #define BLE_GAP_IOCAP_DISPLAY_ONLY |
| --- |
| Display Only iocapability.<br>Output function : Local device has the ability to display a 6 digit decimal number.<br>Input function : None |

### ◆ BLE_GAP_IOCAP_DISPLAY_YESNO

| #define BLE_GAP_IOCAP_DISPLAY_YESNO |
| --- |
| Display Yes/No iocapability.<br><br>Output function : Output function : Local device has the ability to display a 6 digit decimal number.<br>Input function : Local device has the ability to indicate 'yes' or 'no' |

### ◆ BLE_GAP_IOCAP_KEYBOARD_ONLY

| #define BLE_GAP_IOCAP_KEYBOARD_ONLY |
| --- |
| Keyboard Only iocapability.<br><br>Output function : None<br>Input function : Local device has the ability to input the number '0' - '9'. |

### ◆ BLE_GAP_IOCAP_NOINPUT_NOOUTPUT

| #define BLE_GAP_IOCAP_NOINPUT_NOOUTPUT |
| --- |
| No Input No Output iocapability.<br><br>Output function : None<br>Input function : None |

### ◆ BLE_GAP_IOCAP_KEYBOARD_DISPLAY

| #define BLE_GAP_IOCAP_KEYBOARD_DISPLAY |
| --- |
| Keyboard Display iocapability.<br><br>Output function : Output function : Local device has the ability to display a 6 digit decimal number.<br>Input function : Local device has the ability to input the number '0' - '9'. |

## Typedef Documentation

### ◆ ble_gap_app_cb_t

ble_gap_app_cb_t

ble_gap_app_cb_t is the GAP Event callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | event_type | The type of GAP Event. |
| [in] | event_result | The result of API call which generates the GAP Event. |
| [in] | p_event_data | Data notified in the GAP Event. |

**Returns**

none

### ◆ ble_gap_del_bond_cb_t

ble_gap_del_bond_cb_t

ble_gap_del_bond_cb_t is the type of the callback function for delete bonding information stored in non-volatile area.
This type is used in R_BLE_GAP_DeleteBondInfo().

**Parameters**

| | | |
|---|---|---|
| [in] | p_addr | The parameter returns the address of the remote device whose keys are deleted by R_BLE_GAP_DeleteBondInfo(). If R_BLE_GAP_DeleteBondInfo() deletes the keys of all remote devices, the parameter returns NULL. |

**Returns**

none

### ◆ st_ble_gap_adv_param_t

typedef st_ble_gap_ext_adv_param_t st_ble_gap_adv_param_t

Advertising parameters.

**See also**

st_ble_gap_ext_adv_param_t

### ◆ st_ble_gap_scan_param_t

typedef st_ble_gap_ext_scan_param_t st_ble_gap_scan_param_t

Scan parameters.

**See also**
st_ble_gap_ext_scan_param_t

## Enumeration Type Documentation

### ◆ e_ble_gap_evt_t

enum e_ble_gap_evt_t

GAP Event Identifier.

| Enumerator | |
|---|---|
| BLE_GAP_EVENT_INVALID | Invalid GAP Event.<br>**Event Code: 0x1001**<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_STACK_ON | Host Stack has been initialized.<br>When initializing host stack by R_BLE_GAP_Init() has been completed, BLE_GAP_EVENT_STACK_ON event is notified.<br>**Event Code: 0x1002**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_STACK_OFF | Host Stack has been terminated.<br>When terminating host stack by R_BLE_GAP_Terminate() has been completed, BLE_GAP_EVENT_STACK_OFF event is notified.<br>**Event Code: 0x1003**<br>**result:** |

| | |
|---|---|
| | BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_STATE(0x0008) When function was called, host stack has not yet been initialized.<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_LOC_VER_INFO | Version information of local device.<br>When version information of local device has been retrieved by R_BLE_GAP_GetVerInfo(), BLE_GAP_EVENT_LOC_VER_INFO event is notified.<br>**Event Code: 0x1004**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_loc_dev_info_evt_t |
| BLE_GAP_EVENT_HW_ERR | Hardware Error.<br>When hardware error has been received from Controller, BLE_GAP_EVENT_HW_ERR event is notified.<br>**Event Code: 0x1005**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_hw_err_evt_t |
| BLE_GAP_EVENT_CMD_ERR | Command Status Error.<br>When the error of HCI Command has occurred after a R_BLE GAP API call, BLE_GAP_EVENT_CMD_ERR event is notified.<br>**Event Code: 0x1101**<br>**result:** |

| | |
|---|---|
| | BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_cmd_err_evt_t |
| BLE_GAP_EVENT_ADV_REPT_IND | Advertising Report.<br><br>When advertising PDUs has been received after scanning was started by R_BLE_GAP_StartScan().<br><br>**Event Code: 0x1102**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_adv_rept_evt_t |
| BLE_GAP_EVENT_ADV_PARAM_SET_COMP | Advertising parameters have been set.<br><br>Advertising parameters have been configured by R_BLE_GAP_SetAdvParam().<br><br>**Event Code: 0x1103**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_ARG(0x0003) The advertising type that doesn't support advertising data/scan response data was specified to the advertising set which has already set advertising data/scan response data.<br><br>BLE_ERR_INVALID_OPERATION(0x0009) The reason for this error is as follows:<br>Advertising parameters were configured to the advertising set in |

| | | |
|---|---|---|
| | | advertising.<br><br>The sec_adv_phy field in adv_paran was not specified when Periodic Advertising was started. |
| | **Event Data:**<br><br>st_ble_gap_adv_set_evt_t | |
| BLE_GAP_EVENT_ADV_DATA_UPD_COMP | Advertising data has been set.<br><br>This event notifies that Advertising Data/Scan Response Data/Periodic Advertising Data has been set to the advertising set by R_BLE_GAP_SetAdvSresData().<br><br>**Event Code: 0x1104**<br><br>**result:** | |
| | BLE_SUCCESS(0x0000) | Success |
| | BLE_ERR_INVALID_OPERATION(0x0009) | The reason for this error is as follows:<br><br>The advertising set that doesn't support advertising data/scan response data was set to the data.<br><br>The advertising set that supports legacy advertising was set to advertising data/scan response data larger than 31 bytes.<br><br>The advertising set that has advertising data/scan response data greater than or equal to 252 |

<table>
<tr>
<td></td>
<td>bytes was set the data in advertising.<br><br>The advertising set that has periodic advertising data greater than or equal to 253 bytes was set the data in advertising.<br><br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) — Length exceeded the length that the advertising set could be set.<br><br>BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_SetAdvSresData() has not been created.<br><br>Event Data:<br><br>st_ble_gap_adv_data_evt_t</td>
</tr>
<tr>
<td>BLE_GAP_EVENT_ADV_ON</td>
<td>Advertising has started.<br><br>When advertising has been started by R_BLE_GAP_StartAdv(), this event is notified to the application layer.<br><br>Event Code: 0x1105<br><br>result:<br><br>BLE_SUCCESS(0x0000) — Success<br><br>BLE_ERR_INVALID_ARG(0x0003) — The reason for this error is as follows:<br><br>The advertising data length set to the advertising set for connectable extended advertising was invalid.<br><br>If o_addr_type field in adv_param used</td>
</tr>
</table>

| | |
|---|---|
| | in R_BLE_GAP_SetAdvParam() is 0x03, the address which is set in o_addr field of adv_param has not been registered in Resolving List.<br>BLE_ERR_INVALID_OPERATION(0x0009) Setting of advertising data/scan response data has not been completed.<br>BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_StartAdv() has not been created.<br>BLE_ERR_LIMIT_EXCEEDED(0x0010) When the maximum connections are established, a new connectable advertising tried starting.<br>**Event Data:**<br>st_ble_gap_adv_set_evt_t |
| BLE_GAP_EVENT_ADV_OFF | Advertising has stopped.<br>This event notifies the application layer that advertising has stopped.<br>**Event Code: 0x1106**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_StopAdv() has not been created.<br>**Event Data:** |

| | |
|---|---|
| | st_ble_gap_adv_off_evt_t |
| BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP | Periodic advertising parameters have been set.<br>This event notifies the application layer that Periodic Advertising Parameters has been configured by R_BLE_GAP_SetPerdAdvParam().<br>**Event Code: 0x1107**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The advertising set was the setting for anonymous advertising.<br>BLE_ERR_INVALID_OPERATION(0x0009) The advertising set was configured to the parameters in periodic advertising.<br>BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_SetPerdAdvParam() has not been created.<br>**Event Data:**<br>st_ble_gap_adv_set_evt_t |
| BLE_GAP_EVENT_PERD_ADV_ON | Periodic advertising has started.<br>When Periodic Advertising has been started by R_BLE_GAP_StartPerdAdv(), this event is notified to the application layer.<br>**Event Code: 0x1108**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_OPERATION(0x0009) The periodic advertising data set in the advertising set has not been completed. |

| | |
|---|---|
| | BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_StartPerdAdv() has not been created.<br><br>**Event Data:**<br><br>st_ble_gap_adv_set_evt_t |
| BLE_GAP_EVENT_PERD_ADV_OFF | Periodic advertising has stopped.<br><br>When Periodic Advertising has terminated by R_BLE_GAP_StopPerdAdv(), this event is notified to the application layer.<br><br>**Event Code: 0x1109**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_StopPerdAdv() has not been created.<br><br>**Event Data:**<br><br>st_ble_gap_adv_set_evt_t |
| BLE_GAP_EVENT_ADV_SET_REMOVE_COMP | Advertising set has been deleted.<br><br>When the advertising set has been removed by R_BLE_GAP_RemoveAdvSet(), this event is notified to the application layer.<br><br>**Event Code: 0x110A**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_OPERATION(0x0009) When the advertising set was in advertising, R_BLE_GAP_RemoveAdvSet() was called.<br><br>BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_Remov |

| | |
|---|---|
| | eAdvSet() has not been created.<br><br>**Event Data:**<br><br>st_ble_gap_rem_adv_set_evt_t |
| BLE_GAP_EVENT_SCAN_ON | Scanning has started.<br><br>When scanning has started by R_BLE_GAP_StartScan(), this event is notified to the application layer.<br><br>**Event Code: 0x110B**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_ARG(0x0003) The reason for this error is as follows:<br>Scan interval or scan window was invalid.<br>When filter_dup field in scan_enable was BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02), period field in scan_enable was 0.<br>Duration field in scan_enable was larger than period in scan_enable.<br><br>BLE_ERR_INVALID_OPERATION(0x0009) In scanning, R_BLE_GAP_StartScan() was called.<br><br>**Event Data:**<br><br>none |
| BLE_GAP_EVENT_SCAN_OFF | Scanning has stopped.<br><br>When scanning has been stopped by R_BLE_GAP_StopScan(), this event is notified to the application layer. |

| | |
|---|---|
| | **Event Code: 0x110C**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_SCAN_TO | Scanning has stopped, because duration specified by API expired.<br>When the scan duration specified by R_BLE_GAP_StartScan() has expired, this event notifies scanning has stopped.<br>**Event Code: 0x110D**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_CREATE_CONN_COMP | Connection Request has been sent to Controller.<br>This event notifies a request for a connection has been sent to Controller.<br>**Event Code: 0x110E**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The reason for this error is as follows:<br>Scan interval or scan windows specified by R_BLE_GAP_CreateConn() is invalid.<br>Although the own_addr_type field in p_param was set to 0x03, |

| | |
|---|---|
| | random address had not been registered in Resolving List.<br>BLE_ERR_INVALID_OPERATION(0x0009) R_BLE_GAP_CreateConn() was called while creating a link by previous R_BLE_GAP_CreateConn() call .<br>BLE_ERR_LIMIT_EXCEEDED(0x0010) When the maximum connections are established, R_BLE_GAP_CreateConn() was called.<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_CONN_IND | Link has been established.<br>This event notifies a link has been established.<br>**Event Code: 0x110F**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_HDL(0x000E) The request for a connection has been cancelled by R_BLE_GAP_CancelCreateConn().<br>**Event Data:**<br>st_ble_gap_conn_evt_t |
| BLE_GAP_EVENT_DISCONN_IND | Link has been disconnected.<br>This event notifies a link has been disconnected.<br>**Event Code: 0x1110**<br>**result:**<br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| | **Event Data:**<br>st_ble_gap_disconn_evt_t |
| BLE_GAP_EVENT_CONN_CANCEL_COMP | Connection Cancel Request has been sent to Controller.<br>This event notifies the request for a connection has been cancelled by R_BLE_GAP_CancelCreateConn().<br>**Event Code: 0x1111**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_OPERATION(0x0009) When a request for a connection has not been sent to Controller, R_BLE_GAP_CancelCreateConn() was called.<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_WHITE_LIST_CONF_COMP | The White List has been configured.<br>When White List has been configured, this event is notified to the application layer.<br>**Event Code: 0x1112**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_STATE(0x0008) The add or delete operation was called, before the previous clear operation has been completed.<br>BLE_ERR_INVALID_OPERATION(0x0009) While doing advertising or scanning or creating a link with the White List, R_BLE_GAP_ConfWhiteList() was |

| | |
|---|---|
| | called.<br><br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) White List has already registered the maximum number of devices.<br><br>**Event Data:**<br><br>st_ble_gap_white_list_conf_evt_t |
| BLE_GAP_EVENT_RAND_ADDR_SET_COMP | Random address has been set to Controller.<br><br>This event notifies Controller has been set the random address by R_BLE_GAP_SetRandAddr().<br><br>**Event Code: 0x1113**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_OPERATION(0x0009) When local device was in legacy advertising, R_BLE_GAP_SetRandAddr() was called.<br><br>**Event Data:**<br><br>none |
| BLE_GAP_EVENT_CH_MAP_RD_COMP | Channel Map has been retrieved.<br><br>This event notifies Channel Map has been retrieved by R_BLE_GAP_ReadChMap().<br><br>**Event Code: 0x1114**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_HDL(0x000E) The remote device specified by R_BLE_GAP_ReadChMap() was not found.<br><br>**Event Data:**<br><br>st_ble_gap_rd_ch_map_evt_t |

<table>
<tr><td>BLE_GAP_EVENT_CH_MAP_SET_COMP</td><td>Channel Map has set.<br><br>This event notifies Channel Map has been configured by R_BLE_GAP_SetChMap().<br><br>Event Code: 0x1115<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_ARG(0x0003) The channel map specified by R_BLE_GAP_SetChMap() was all-zero.<br><br>Event Data:<br><br>none</td></tr>
<tr><td>BLE_GAP_EVENT_RSSI_RD_COMP</td><td>RSSI has been retrieved.<br><br>This event notifies RSSI has been retrieved by R_BLE_GAP_ReadRssi().<br><br>Event Code: 0x1116<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_HDL(0x000E) The remote device specified by R_BLE_GAP_ReadRssi() was not found.<br><br>Event Data:<br><br>st_ble_gap_rd_rssi_evt_t</td></tr>
<tr><td>BLE_GAP_EVENT_GET_REM_DEV_INFO</td><td>Information about the remote device has been retrieved.<br><br>This event notifies information about the remote device has been retrieved by R_BLE_GAP_GetRemDevInfo().<br><br>Event Code: 0x1117<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success</td></tr>
</table>

| | |
|---|---|
| | **Event Data:**<br>st_ble_gap_dev_info_evt_t |
| BLE_GAP_EVENT_CONN_PARAM_UPD_COMP | Connection parameters has been configured.<br>This event notifies the connection parameters has been updated.<br>**Event Code: 0x1118**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_DATA(0x0002) Local device rejected the request for updating connection parameters.<br>BLE_ERR_INVALID_ARG(0x0003) The remote device rejected the connection parameters suggested from local device.<br>BLE_ERR_UNSUPPORTED(0x0007) The remote device doesn't support connection parameters update feature.<br>**Event Data:**<br>st_ble_gap_conn_upd_evt_t |
| BLE_GAP_EVENT_CONN_PARAM_UPD_REQ | Local device has received the request for configuration of connection parameters.<br>This event notifies the request for connection parameters update has been received.<br>**Event Code: 0x1119**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:** |

| | |
|---|---|
| | st_ble_gap_conn_upd_req_evt_t |
| BLE_GAP_EVENT_AUTH_PL_TO_EXPIRED | Authenticated Payload Timeout.<br>This event notifies Authenticated Payload Timeout has occurred.<br>**Event Code: 0x111A**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_SET_DATA_LEN_COMP | The request for update transmission packet size and transmission time have been sent to Controller.<br>This event notifies a request for updating packet data length and transmission timer has been sent to Controller.<br>**Event Code: 0x111B**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The tx_octets or tx_time parameter specified by R_BLE_GAP_SetDataLen() is invalid.<br>BLE_ERR_UNSUPPORTED(0x0007) The remote device does not support updating packet data length and transmission time.<br>BLE_ERR_INVALID_HDL(0x000E) When R_BLE_GAP_SetDataLen() was called, the connection was not established.<br>**Event Data:**<br>st_ble_gap_conn_hdl_evt_t |

<table>
<tr>
<td>BLE_GAP_EVENT_DATA_LEN_CHG</td>
<td>Transmission packet size and transmission time have been changed.<br><br>This event notifies packet data length and transmission time have been updated.<br><br>Event Code: 0x111C<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>Event Data:<br><br>st_ble_gap_data_len_chg_evt_t</td>
</tr>
<tr>
<td>BLE_GAP_EVENT_RSLV_LIST_CONF_COMP</td>
<td>The Resolving List has been configured.<br><br>When Resolving List has been configured by R_BLE_GAP_ConfRslvList(), this event is notified to the application layer.<br><br>Event Code: 0x111D<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_STATE(0x0008) The add or delete operation was called, before the previous clear operation has been completed.<br><br>BLE_ERR_INVALID_OPERATION(0x0009) While doing advertising or scanning or creating a link with resolvable private address, R_BLE_GAP_ConfRslvList() was called.<br><br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) Resolving List has already registered the maximum number of devices.<br><br>BLE_ERR_INVALID_HDL(0x000E) The specified Identity Address was not found in Resolving List.</td>
</tr>
</table>

| | |
|---|---|
| | **Event Data:**<br><br>st_ble_gap_rslv_list_conf_evt_t |
| BLE_GAP_EVENT_RPA_EN_COMP | Resolvable private address function has been enabled or disabled.<br><br>When Resolvable Private Address function in Controller has been enabled by R_BLE_GAP_EnableRpa(), this event is notified to the application layer.<br><br>**Event Code: 0x111E**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_OPERATION(0x0009) While advertising, scanning, or establishing a link with resolvable private address, R_BLE_GAP_EnableRpa() was called.<br><br>**Event Data:**<br><br>none |
| BLE_GAP_EVENT_SET_RPA_TO_COMP | The update time of resolvable private address has been changed.<br><br>When Resolvable Private Address Timeout in Controller has been updated by R_BLE_GAP_SetRpaTo(), this event is notified to the application layer.<br><br>**Event Code: 0x111F**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_ARG(0x0003) The rpa_timeout parameter specified by R_BLE_GAP_SetRpaTo() is out of range.<br><br>**Event Data:**<br><br>none |

<table>
<tr><td>BLE_GAP_EVENT_RD_RPA_COMP</td><td>The resolvable private address of local device has been retrieved.<br><br>When the resolvable private address of local device has been retrieved by R_BLE_GAP_ReadRpa(), this event is notified to the application layer.<br><br>Event Code: 0x1120<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_HDL(0x000E) The identity address specified by R_BLE_GAP_ReadRpa() was not registered in Resolving List.<br><br>Event Data:<br><br>st_ble_gap_rd_rpa_evt_t</td></tr>
<tr><td>BLE_GAP_EVENT_PHY_UPD</td><td>PHY for connection has been changed.<br><br>This event notifies the application layer that PHY for a connection has been updated.<br><br>Event Code: 0x1121<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>Event Data:<br><br>st_ble_gap_phy_upd_evt_t</td></tr>
<tr><td>BLE_GAP_EVENT_PHY_SET_COMP</td><td>The request for updating PHY for connection has been sent to Controller.<br><br>When Controller has received a request for updating PHY for a connection by R_BLE_GAP_SetPhy(), this event is notified to the application layer.<br><br>Event Code: 0x1122<br><br>result:</td></tr>
</table>

| | |
|---|---|
| | BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_HDL(0x000E) The remote device specified by R_BLE_GAP_SetPhy() was not found.<br>**Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_DEF_PHY_SET_COMP | The request for setting default PHY has been sent to Controller.<br>When the PHY preferences which a remote device may change has been configured by R_BLE_GAP_SetDefPhy(), this event is notified to the application layer.<br>**Event Code: 0x1123**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_PHY_RD_COMP | PHY configuration has been retrieved.<br>When the PHY settings has been retrieved by R_BLE_GAP_ReadPhy(), this event is notified to the application layer.<br>**Event Code: 0x1124**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_HDL(0x000E) The link specified by R_BLE_GAP_ReadPhy() was not found.<br>**Event Data:**<br>st_ble_gap_phy_rd_evt_t |
| BLE_GAP_EVENT_SCAN_REQ_RECV | Scan Request has been received. |

| | |
|---|---|
| | This event notifies the application layer that a Scan Request packet has been received from a Scanner.<br>**Event Code: 0x1125**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_scan_req_recv_evt_t |
| BLE_GAP_EVENT_CREATE_SYNC_COMP | The request for establishing a periodic sync has been sent to Controller.<br>This event notifies the application layer that Controller has received a request for a Periodic Sync establishment.<br>**Event Code: 0x1126**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_OPERATION(0x0009) When R_BLE_GAP_CreateSync() was called, this event for previous the API call has not been received.<br>BLE_ERR_ALREADY_IN_PROGRESS(0x000A) The advertising set specified by R_BLE_GAP_CreateSync() has already established a periodic sync.<br>**Event Data:**<br>none |
| BLE_GAP_EVENT_SYNC_EST | The periodic advertising sync has been established.<br>This event notifies the application layer that a Periodic sync has been established.<br>**Event Code: 0x1127** |

| | |
|---|---|
| | **result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_NOT_YET_READY(0x0012) The request for a Periodic Sync establishment was cancelled by R_BLE_GAP_CancelCreateSync().<br>**Event Data:**<br>st_ble_gap_sync_est_evt_t |
| BLE_GAP_EVENT_SYNC_TERM | The periodic advertising sync has been terminated.<br>This event notifies the application layer that the Periodic Sync has been terminated by R_BLE_GAP_TerminateSync().<br>**Event Code: 0x1128**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_OPERATION(0x0009) While establishing a Periodic Sync by R_BLE_GAP_CreateSync(), R_BLE_GAP_TerminateSync() was called.<br>BLE_ERR_INVALID_HDL(0x000E) The sync handle specified by R_BLE_GAP_TerminateSync() was not found.<br>**Event Data:**<br>st_ble_gap_sync_hdl_evt_t |
| BLE_GAP_EVENT_SYNC_LOST | The periodic advertising sync has been lost.<br>This event notifies the application layer that the Periodic Sync has been lost.<br>**Event Code: 0x1129**<br>**result:** |

| | |
|---|---|
| | BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_sync_hdl_evt_t |
| BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP | The request for cancel of establishing a periodic advertising sync has been sent to Controller.<br><br>This event notifies the request for a Periodic Sync establishment has been cancelled by R_BLE_GAP_CancelCreateSync().<br><br>**Event Code: 0x112A**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_OPERATION(0x0009) When R_BLE_GAP_CancelCreateSync() was called, a request for a Periodic Sync establishment by R_BLE_GAP_CreateSync() has not been sent to Controller.<br><br>**Event Data:**<br><br>none |
| BLE_GAP_EVENT_PERD_LIST_CONF_COMP | The Periodic Advertiser list has been configured.<br><br>When Periodic Advertiser List has been configured by R_BLE_GAP_ConfPerdAdvList(), this event is notified to the application layer.<br><br>**Event Code: 0x112B**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_INVALID_ARG(0x0003) The advertiser has already been registered in Periodic Advertiser |

| | |
|---|---|
| | List.<br>BLE_ERR_INVALID_STATE(0x0008) The add or delete operation was called, before the previous clear operation has been completed.<br>BLE_ERR_INVALID_OPERATION(0x0009) When establishing a periodic sync by R_BLE_GAP_CreateSync(), R_BLE_GAP_ConfPerdAdvList() was called.<br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) Periodic Advertiser List has already registered the maximum number of devices.<br>BLE_ERR_INVALID_HDL(0x000E) The device specified by R_BLE_GAP_ConfPerdAdvList() was not found.<br>**Event Data:**<br>st_ble_gap_perd_list_conf_evt_t |
| BLE_GAP_EVENT_PRIV_MODE_SET_COMP | Privacy Mode has been configured.<br>This event notifies the application layer that the Privacy Mode has been configured by R_BLE_GAP_SetPrivMode().<br>**Event Code: 0x112B**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) Address type or privacy mode is out of range.<br>BLE_ERR_INVALID_OPERATION(0x0009) While advertising, scanning, or establishing a link with resolvable private address, R_BLE_GAP_SetPrivMode() was |

| | |
|---|---|
| | called.<br><br>BLE_ERR_INVALID_HDL(0x000E) The address specified by R_BLE_GAP_SetPrivMode() has not been registered in Resolving List.<br><br>**Event Data:**<br><br>none |
| BLE_GAP_EVENT_PAIRING_REQ | The pairing request from a remote device has been received.<br><br>This event notifies the application layer that a pairing request from a remote device has been received.<br><br>**Event Code: 0x1401**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_pairing_info_evt_t |
| BLE_GAP_EVENT_PASSKEY_ENTRY_REQ | The request for input passkey has been received.<br><br>This event notifies that a request for Passkey input in pairing has been received.<br><br>**Event Code: 0x1402**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ | The request for displaying a passkey has been received.<br><br>This event notifies that a request for Passkey display in pairing has been received.<br><br>**Event Code: 0x1403** |

| | |
|---|---|
| | **result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_passkey_display_evt_t |
| BLE_GAP_EVENT_NUM_COMP_REQ | The request for confirmation with Numeric Comparison has received.<br>This event notifies that a request for Numeric Comparison in pairing has been received.<br>**Event Code: 0x1404**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_num_comp_evt_t |
| BLE_GAP_EVENT_KEY_PRESS_NTF | Key Notification from a remote device has been received.<br>This event notifies the application layer that the remote device has input a key in Passkey Entry.<br>**Event Code: 0x1405**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gap_key_press_ntf_evt_t |
| BLE_GAP_EVENT_PAIRING_COMP | Pairing has been completed.<br>This event notifies the application layer that the pairing has completed.<br>**Event Code: 0x1406**<br>**result:**<br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| BLE_ERR_SMP_LE_PASSKEY_ENTRY_FAIL(0x2001) | PassKey Entry is failed. |
| BLE_ERR_SMP_LE_OOB_DATA_NOT_AVAILABLE(0x2002) | OOB Data is not available. |
| BLE_ERR_SMP_LE_AUTH_REQ_NOT_MET(0x2003) | The requested pairing can not be performed because of IO Capability. |
| BLE_ERR_SMP_LE_CONFIRM_VAL_NOT_MATCH(0x2004) | Confirmation value does not match. |
| BLE_ERR_SMP_LE_PAIRING_NOT_SPRT(0x2005) | Pairing is not supported. |
| BLE_ERR_SMP_LE_INSUFFICIENT_ENC_KEY_SIZE(0x2006) | Encryption Key Size is insufficient. |
| BLE_ERR_SMP_LE_CMD_NOT_SPRT(0x2007) | The pairing command received is not supported. |
| BLE_ERR_SMP_LE_UNSPECIFIED_REASON(0x2008) | Pairing failed with an unspecified reason. |
| BLE_ERR_SMP_LE_REPEATED_ATTEMPTS(0x2009) | The number of repetition exceeded the upper limit. |
| BLE_ERR_SMP_LE_INVALID_PARAM(0x200A) | Invalid parameter is set. |
| BLE_ERR_SMP_LE_DHKEY_CHECK_FAIL(0x200B) | DHKey Check error. |
| BLE_ERR_SMP_LE_NUM_COMP_FAIL(0x200C) | Numeric Comparison failure. |
| BLE_ERR_SMP_LE_DISCONNECTED(0x200F) | Disconnection in pairing. |
| BLE_ERR_SMP_LE_TO(0x2011) | Failure due to timeout. |
| BLE_ERR_SMP_LE_LOC_KEY_MISSING( | Pairing/Encryption failure because |

| | |
|---|---|
| | 0x2014) local device lost the LTK.<br><br>**Event Data:**<br><br>st_ble_gap_pairing_info_evt_t |
| BLE_GAP_EVENT_ENC_CHG | Key Notification from a remote device has been received.<br><br>This event notifies the application layer that the encryption status of a link has been changed.<br><br>**Event Code: 0x1407**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_enc_chg_evt_t |
| BLE_GAP_EVENT_PEER_KEY_INFO | Keys has been received from a remote device.<br><br>This event notifies the application layer that the remote device has distributed the keys.<br><br>**Event Code: 0x1408**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_peer_key_info_evt_t |
| BLE_GAP_EVENT_EX_KEY_REQ | The request for key distribution has been received.<br><br>When local device has been received a request for key distribution to remote device, this event is notified to the application layer.<br><br>**Event Code: 0x1409**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| | **Event Data:**<br><br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_LTK_REQ | LTK has been request from a remote device.<br><br>When local device has been received a LTK request from a remote device, this event is notified to the application layer.<br><br>**Event Code: 0x140A**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_ltk_req_evt_t |
| BLE_GAP_EVENT_LTK_RSP_COMP | LTK reply has been sent to Controller.<br><br>When local device has replied to the LTK request from the remote device, this event is notified to the application layer.<br><br>**Event Code: 0x140B**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_ltk_rsp_evt_t |
| BLE_GAP_EVENT_SC_OOB_CREATE_COMP | The authentication data to be used in Secure Connections OOB has been created.<br><br>This event notifies OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData().<br><br>**Event Code: 0x140C**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gap_sc_oob_data_evt_t |

| | |
|---|---|
| BLE_GAP_EVENT_CTE_CONN_REQ_FAILED | An connectionless CTE IQ sample is reported.<br>**result**<br>0x0000 Response without CTE info other Rejected by remote peer<br>**Event Data:**<br>None |
| BLE_GAP_EVENT_CTE_CONNLESS_REPT | An connectionless CTE IQ sample is reported.<br>**Event Data:**<br>st_ble_gap_cte_connless_rept_t |
| BLE_GAP_EVENT_CTE_CONN_REPT | An connection CTE IQ sample is reported.<br>**Event Data:**<br>st_ble_gap_cte_conn_rept_t |
| BLE_GAP_EVENT_SUBRATE_CHANGE | a Connection Subrate Update procedure has completed and some parameters of the specified connection have changed.<br>**Event Data:**<br>st_ble_subrate_upd_t |
| BLE_GAP_EVENT_PAST_RECV | This event notifies that it has received periodic advertising synchronization information from the device referred to by the Connection_Handle parameter<br>**Event Data:**<br>st_ble_gap_past_est_evt_t. |
| BLE_GAP_EVENT_TX_POWER_REPT | Transmit power level report.<br>This event is a report of the transmit power level on the ACL connection identified by the conn_hdl<br>**Event Data:**<br>st_ble_gap_tx_power_reporting_evt_t |
| BLE_GAP_EVENT_PATH_LOSS_THR | Report a path loss threshold crossing on the ACL connection identified by the Connection_Handle parameter.<br>**Event Data:** |

| | |
|---|---|
| | st_ble_gap_pass_loss_thr_evt_t |
| BLE_GAP_EVENT_REQ_PEER_SCA_COMP | Indicates that the HCI_LE_Request_Peer_SCA command has been completed.<br>**Event Data:**<br>st_ble_gap_req_peer_sca_evt_t. |
| BLE_GAP_EVENT_CTE_SET_CONNLESS_PARAM_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_CTE_CONNLESS_TX_ON | **Event Data:**<br>None |
| BLE_GAP_EVENT_CTE_CONNLESS_TX_OFF | **Event Data:**<br>None |
| BLE_GAP_EVENT_CTE_CONNLESS_RX_ON | **Event Data:**<br>st_ble_gap_sync_hdl_evt_t |
| BLE_GAP_EVENT_CTE_CONNLESS_RX_OFF | **Event Data:**<br>st_ble_gap_sync_hdl_evt_t |
| BLE_GAP_EVENT_CTE_SET_CONN_PARAM_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_CTE_SET_CONN_RSP_ON | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_CTE_SET_CONN_RSP_OFF | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_CTE_SET_CONN_RECV_PARAM_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_CTE_CONN_REQ_ON | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_CTE_CONN_REQ_OFF | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_SET_DEF_SUBRATE_COMP | **Event Data:**<br>None |

| | |
|---|---|
| BLE_GAP_EVENT_REQ_SUBRATE_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_PAST_START_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PAST_SET_PARAM_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PAST_SET_DEF_PARAM_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_UPD_SCA_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_READ_REMOTE_TX_POWER_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_SET_PATHLOSS_REPT_PARAM_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PATHLOSS_REPT_ON | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PATHLOSS_REPT_OFF | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_LOCAL_TX_POWER_REPT_ON | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_LOCAL_TX_POWER_REPT_OFF | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_REMOTE_TX_POWER_REPT_ON | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_REMOTE_TX_POWER_REPT_OFF | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_SET_RPA_UPD_REASON_COMP | **Event Data:**<br>None |
| | |

| | |
|---|---|
| BLE_GAP_EVENT_DTM_RX_TEST_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_DTM_TX_TEST_COMP | **Event Data:**<br>None |
| BLE_GAP_EVENT_DTM_TEST_END_COMP | **Event Data:**<br>st_ble_gap_dtm_test_end_evt_t |
| BLE_GAP_EVENT_ENHANCED_READ_TX_POWER_LEVEL_COMP | **Event Data:**<br>st_ble_gap_enhanced_read_tx_power_level_evt_t |
| BLE_GAP_EVENT_SET_HOST_FEAT_COMP | **Event Data:**<br>None |

## Function Documentation

### ◆ R_BLE_GAP_Init()

ble_status_t R_BLE_GAP_Init ( ble_gap_app_cb_t *gap_cb*)

Initialize the Host Stack.

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback function is registered with this function. In order to receive the GAP event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event.

**Parameters**

| | | |
|---|---|---|
| [in] | gap_cb | A callback function registered with this function. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | gap_cb is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• Host Stack was already initialized.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_Terminate()

ble_status_t R_BLE_GAP_Terminate ( void )

Terminate the Host Stack.

Host stack is terminated with this function. In order to reset all the Bluetooth functions, it's necessary to call this function. The result of this API call is notified in BLE_GAP_EVENT_STACK_OFF event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | Host stack hasn't been initialized. |

### ◆ R_BLE_GAP_UpdConn()

ble_status_t R_BLE_GAP_UpdConn ( uint16_t *conn_hdl*, uint8_t *mode*, uint16_t *accept*, st_ble_gap_conn_param_t * *p_conn_updt_param* )

Update the connection parameters.

This function updates the connection parameters or replies a request for updating connection parameters notified by BLE_GAP_EVENT_CONN_PARAM_UPD_REQ event. When the connection parameters has been updated, BLE_GAP_EVENT_CONN_PARAM_UPD_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link to be updated. |
| [in] | mode | Connection parameter update request or response.<br>macro: BLE_GAP_CONN_UPD_MODE_REQ (0x01) — description: Request for updating the connection parameters.<br>macro: BLE_GAP_CONN_UPD_MODE_RSP (0x02) — description: Reply a connection parameter update request. |
| [in] | accept | When mode is BLE_GAP_CONN_UPD_MODE_RSP, accept or reject the connection parameters update request. If mode is BLE_GAP_CONN_U |

| | | |
|---|---|---|
| | | PD_MODE_REQ, accept is ignored.<br>macro: description<br>BLE_GAP_CONN_UPD_ACCEPT (0x0000): Accept the update request.<br>BLE_GAP_CONN_UPD_REJECT (0x0001): Reject the update request. |
| [in] | p_conn_updt_param | Connection parameters to be updated. When mode is BLE_GAP_CONN_UPD_MODE_RSP and accept is BLE_GAP_CONN_UPD_REJECT, p_conn_updt_param is ignored. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | When accept is BLE_GAP_CONN_UPD_ACCEPT, p_conn_updt_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The following is out of range.<br>• mode<br>• accept<br>• conn_intv_min field in p_conn_updt_param<br>• conn_intv_max field in p_conn_updt_param<br>• conn_latency in p_conn_updt_param<br>• sup_to in p_conn_updt_param<br>• conn_hdl |
| BLE_ERR_INVALID_STATE(0x0008) | Not connected with the remote device. |
| BLE_ERR_CONTEXT_FULL(0x000B) | Sending a L2CAP command, an error occurred. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_SetDataLen()

ble_status_t R_BLE_GAP_SetDataLen ( uint16_t *conn_hdl*, uint16_t *tx_octets*, uint16_t *tx_time* )

Update the packet size and the packet transmit time.

This function requests for changing the maximum transmission packet size and the maximum packet transmission time. When Controller has received the request from host stack, BLE_GAP_EVENT_SET_DATA_LEN_COMP event is notified to the application layer. When the transmission packet size or the transmission time has been changed, BLE_GAP_EVENT_DATA_LEN_CHG event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose the transmission packet size or the transmission time to be changed. |
| [in] | tx_octets | Maximum transmission packet size. Valid range is 0x001B - 0x00FB. |
| [in] | tx_time | Maximum transmission time(us). Valid range is 0x0148 - 0x4290. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_Disconnect()

ble_status_t R_BLE_GAP_Disconnect ( uint16_t *conn_hdl*, uint8_t *reason* )

Disconnect the link.

This function disconnects a link. When the link has disconnected, BLE_GAP_EVENT_DISCONN_IND event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link to be disconnected. |
| [in] | reason | The reason for disconnection. Usually, set 0x13 which indicates that a user disconnects the link. If setting other than 0x13, refer the error code described in Core Specification Vol.2 Part D ,"2 Error Code Descriptions". |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | conn_hdl is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_SetPhy()

ble_status_t R_BLE_GAP_SetPhy ( uint16_t *conn_hdl*, st_ble_gap_set_phy_param_t * *p_phy_param* )

Set the phy for connection.

This function sets the PHY preferences for the connection. The result of this API call is notified in BLE_GAP_EVENT_PHY_SET_COMP event. When the PHY has been updated, BLE_GAP_EVENT_PHY_UPD event is notified to the application layer.

After PHY update, the PHY accept configuration of local device is the same as the values in BLE_GAP_EVENT_PHY_UPD event.
For example, after calling R_BLE_GAP_SetPhy(), if tx_phy, rx_phy by BLE_GAP_EVENT_PHY_UPD event are updated to 2M PHY, the PHY accept configuration is 2M PHY only.
Therefore after receiving BLE_GAP_EVENT_PHY_UPD event, if local device wants to accept the other PHY configuration, it needs to call R_BLE_GAP_SetPhy() with the desired PHY accept configuration.

Because the maximum transmission packet size or the maximum transmission time might be updated by PHY update, if the same packet size or transmission time as the previous one is desired, change the maximum transmission packet size or the maximum transmission time by R_BLE_GAP_SetDataLen().

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose PHY to be updated. |
| [in] | p_phy_param | PHY preferences. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_phy_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | conn_hdl or option field in p_phy_param is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetDefPhy()

ble_status_t R_BLE_GAP_SetDefPhy ( st_ble_gap_set_def_phy_param_t * *p_def_phy_param*)

Set the default phy which allows remote device to change.

This function sets the PHY preferences which a remote device may change. The result of this API call is notified in BLE_GAP_EVENT_DEF_PHY_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_def_phy_param | The PHY preference which a remote device may change. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_def_phy_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | tx_phys or tx_phys field in p_def_phy_param is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetPrivMode()

ble_status_t R_BLE_GAP_SetPrivMode ( st_ble_dev_addr_t * *p_addr*, uint8_t * *p_privacy_mode*, uint8_t *device_num* )

Set the privacy mode.

This function sets privacy mode for the remote device registered in Resolving List. By default, Network Privacy Mode is set.
The result of this API call is notified in BLE_GAP_EVENT_PRIV_MODE_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_addr | An array of identity address of the remote device to set privacy mode. The number of elements is specified by device_num. |
| [in] | p_privacy_mode | An array of privacy mode to set to remote device. The number of elements is specified by device_num. The following value is set as |

| | | |
|---|---|---|
| | | the privacy mode.<br>macro: BLE_GAP_NET_PRIV_MODE (0x00) — Network Privacy Mode.<br>BLE_GAP_DEV_PRIV_MODE (0x01) — Device Privacy Mode. |
| [in] | device_num | The number of devices to set privacy mode. Valid range is 1-BLE_GAP_RSLV_LIST_MAX_ENTRY. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_addr or p_privacy_mode is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The following parameter is out of range.<br>• The address type in p_addr.<br>• The privacy mode specified by p_privacy_mode.<br>• device_num |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• While configuring privacy mode, this function was called.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ConfWhiteList()

ble_status_t R_BLE_GAP_ConfWhiteList ( uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, uint8_t *device_num* )

Set White List.

This function supports the following operations regarding White List.

- Add the device to White List.
- Delete the device from White List.

- Clear White List.

  The total number of White List entries is defined as BLE_GAP_WHITE_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_WHITE_LIST_CONF_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | op_code | The operation for White List.<br>macro: BLE_GAP_LIST_ADD_DEV(0x01) — Add the device to the list.<br>BLE_GAP_LIST_REM_DEV(0x02) — Delete the device from the list.<br>BLE_GAP_LIST_CLR(0x03) — Clear the list. |
| [in] | p_addr | An array of device address to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored. |
| [in] | device_num | The number of devices add / delete to the list. Valid range is 1-BLE_GAP_WHITE_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

| | |
|---|---|
| BLE_ERR_INVALID_PTR(0x0001) | When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, p_addr is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | op_code or address type field in p_addr is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• While operating White List, this function was called.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for operating the White List. |

### ◆ R_BLE_GAP_GetVerInfo()

ble_status_t R_BLE_GAP_GetVerInfo ( void )

Get the version number of the Controller and the host stack.

This function retrieves the version information of local device. The result of this API call is notified in BLE_GAP_EVENT_LOC_VER_INFO event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ReadPhy()

ble_status_t R_BLE_GAP_ReadPhy ( uint16_t *conn_hdl*)

Get the phy settings.

This function gets the PHY settings for the connection. The result of this API call is notified in BLE_GAP_EVENT_PHY_RD_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose PHY settings to be retrieved. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | conn_hdl is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ConfRslvList()

ble_status_t R_BLE_GAP_ConfRslvList ( uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, st_ble_gap_rslv_list_key_set_t * *p_peer_irk*, uint8_t *device_num* )

Set Resolving List.

This function supports the following operations regarding Resolving List.

- Add the device to Resolving List.
- Delete the device from Resolving List.

- Clear Resolving List.

In order to generate a resolvable private address, a local IRK needs to be registered by R_BLE_GAP_SetLocIdInfo(). If communicating with the identity address, register all-zero IRK as local IRK. In order to resolve resolvable private address of the remote device, the IRK distributed from the remote device needs to be added to Resolving List. The total number of Resolving List entries is defined as BLE_GAP_RESOLV_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | op_code | The operation for Resolving List.<br>macro: description<br>BLE_GAP_LIST_ADD_DEV(0x01): Add the device to the list.<br>BLE_GAP_LIST_REM_DEV(0x02): Delete the device from the list.<br>BLE_GAP_LIST_CLR(0x03): Clear the list. |
| [in] | p_addr | An array of Identity Addresses to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored. |
| [in] | p_peer_irk | The remote IRK and the type of local IRK added to Resolving List. If op_code is other than BLE_GAP_LIST_ADD_DEV, p_peer_irk is ignored. The number of elements is |

| | | |
|---|---|---|
| | | specified by device_num. |
| [in] | device_num | The number of devices add / delete to the list. Valid range is 1-BLE_GAP_RSLV_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• When added to or deleted from the list, p_addr is specified as NULL.<br>• When added to the list, p_peer_irk is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• op_code is out of range.<br>• When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, device_num is out of range.<br>• When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, address type field in p_addr is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• While operating Resolving List, this function was called.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for operating the Resolving List. |
| BLE_ERR_INVALID_HDL(0x000E) | The specified Identity Address was not found in Resolving List. |

### ◆ R_BLE_GAP_EnableRpa()

ble_status_t R_BLE_GAP_EnableRpa ( uint8_t *enable*)

Enable/Disable address resolution and generation of a resolvable private address.

This function enables or disables RPA functionality. The RPA functionality includes the following.

- Generation of local resolvable private address
- Resolution of remote resolvable private address

In order to do advertising, scanning or creating a link with local resolvable private address, the RPA functionality needs to be enabled. After enabling the RPA functionality and the identity address of remote device and the IRKs of local/remote device is registered, local device can generate own resolvable private address in the time interval set by R_BLE_GAP_SetRpaTo(), and can resolve a resolvable private address of a remote device. It is recommended that the RPA functionality is called immediately after the initialization by R_BLE_GAP_Init(). The result of this API call is notified in BLE_GAP_EVENT_RPA_EN_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | enable | Enable or disable address resolution function.<br>macro: description<br>BLE_GAP_RPA_DISABLED (0x00): Disable RPA generation/resolution.<br>BLE_GAP_RPA_ENABLED( 0x01): Enable RPA generation/resolution. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | enable is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetRpaTo()

ble_status_t R_BLE_GAP_SetRpaTo ( uint16_t *rpa_timeout*)

Set the update time of resolvable private address.

This function sets the time interval to update the resolvable private address. The result of this API call is notified in BLE_GAP_EVENT_SET_RPA_TO_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | rpa_timeout | Time interval to update resolvable private address in seconds. Valid range is 0x003C - 0xA1B8. Default is 900s. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ReadRpa()

ble_status_t R_BLE_GAP_ReadRpa ( st_ble_dev_addr_t * *p_addr*)

Get the resolvable private address of local device.

This function retrieves the local resolvable private address. Before getting the address, enable the resolvable private address function by R_BLE_GAP_EnableRpa(). The result of this API call is notified in BLE_GAP_EVENT_RD_RPA_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_addr | Identity address registered in Resolving List. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_addr is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | Address type in p_addr is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• When retrieving the local resolvable private address, this function was called.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ReadRssi()

ble_status_t R_BLE_GAP_ReadRssi ( uint16_t *conn_hdl*)

Get RSSI.

This function retrieves RSSI. The result of this API call is notified in BLE_GAP_EVENT_RSSI_RD_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose RSSI to be retrieved. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | conn_hdl is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ReadChMap()

ble_status_t R_BLE_GAP_ReadChMap ( uint16_t *conn_hdl*)

Get the Channel Map.

This function retrieves the channel map. The result of this API call is notified in BLE_GAP_EVENT_CH_MAP_RD_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose channel map to be retrieved. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | conn_hdl is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetRandAddr()

ble_status_t R_BLE_GAP_SetRandAddr ( uint8_t * *p_random_addr*)

Set a random address.

This function sets static address or non-resolvable private address to Controller. Refer to Core Specification Vol 6, PartB, "1.3.2 Random Device Address" regarding the format of the random address. Resolvable private address cannot set by this API. The result of this API call is notified in BLE_GAP_EVENT_RAND_ADDR_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_random_addr | Static address or non-resolvable private address. The BD address setting format is little endian. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_random_addr is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetAdvParam()

ble_status_t R_BLE_GAP_SetAdvParam ( st_ble_gap_adv_param_t * *p_adv_param*)

Set advertising parameters.

This function sets advertising parameters. It's possible to do advertising where the advertising parameters are different every each advertising set. The number of advertising set in the Controller is defined as BLE_MAX_NO_OF_ADV_SETS_SUPPORTED. Each advertising set is identified with advertising handle (0x00-0x03). Create an advertising set with this function before start advertising, setting periodic advertising parameters, start periodic advertising, setting advertising data/scan response data/periodic advertising data. The result of this API call is notified in BLE_GAP_EVENT_ADV_PARAM_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_adv_param | Advertising parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_adv_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The below p_adv_param field value is out of range.<br>• adv_handle<br>• adv_intv_min/adv_intv_max<br>• adv_ch_map<br>• o_addr_type<br>• p_addr_type<br>• adv_phy<br>• sec_adv_phy<br>• scan_req_ntf_flag |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetAdvSresData()

ble_status_t R_BLE_GAP_SetAdvSresData ( st_ble_gap_adv_data_t * *p_adv_srsp_data*)

Set advertising data/scan response data/periodic advertising data.

This function sets advertising data/scan response data/periodic advertising data to the advertising set. It is necessary to create an advertising set by R_BLE_GAP_SetAdvParam(), before calling this function. Set advertising data/scan response data/periodic advertising data, after allocating the memory for the data. The following shall be applied regarding the adv_prop_type field and the data_type field in st_ble_gap_adv_param_t parameter specified in R_BLE_GAP_SetAdvParam().

The following shall be applied regarding the adv_prop_type field and the data_type field in st_ble_gap_adv_param_t parameter specified in R_BLE_GAP_SetAdvParam().

- When adv_prop_type is Legacy Advertising PDU type,
  - it's possible to set advertising data/scan response data up to 31 bytes.
  - advertising data/scan response data can be updated by this function in advertising.
- When adv_prop_type is Extended Advertising PDU type,
  - it's possible to set at most 1650 bytes of data as advertising data/scan response data per 1 advertising set.
  - the total buffer size in Controller for advertising data/scan response data is 4250 bytes. Therefore please note that more than 4250 bytes of advertising data/scan response data can not be set to all the advertising sets. Please refer to Figure 1.1 and Figure 1.2 about examples of setting advertising data/scan response data.
  - it's possible to update advertising data/scan response data in advertising, if the data_length field in st_ble_gap_adv_data_t parameter is up to 251 bytes.

adv_data_alloc_fail_en.svg

Figure 349: Figure 1.1

adv_data_alloc_success_en.svg
Figure 350: Figure 1.2

- When periodic advertising data is set,
  - At most 1650 bytes of data can be set to 1 advertising set.
  - The total buffer size in Controller for periodic advertising data is 4306 bytes. Therefore please note that more than 4306 bytes of periodic advertising data can not be set to all the advertising sets.
  - it's possible to update periodic advertising data in advertising, if the data_length field in st_ble_gap_adv_data_t parameter is up to 252 bytes.

The result of this API call is notified in BLE_GAP_EVENT_ADV_DATA_UPD_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_adv_srsp_data | Advertising data/scan response data/periodic advertising data. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• p_adv_srsp_data is specified as NULL.<br>• data_length field in p_adv_srsp_data parameter is not 0 and p_data field is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The following field in p_adv_srsp_data parameter is out of range.<br>• adv_hdl<br>• data_type<br>• data_length<br>• zero_length_flag |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StartAdv()

ble_status_t R_BLE_GAP_StartAdv ( uint8_t *adv_hdl*, uint16_t *duration*, uint8_t *max_extd_adv_evts* )

Start advertising.

This function starts advertising. Create the advertising set specified with adv_hdl by R_BLE_GAP_SetAdvParam(), before calling this function. The result of this API call is notified in BLE_GAP_EVENT_ADV_ON event.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | The advertising handle pointing to the advertising set which starts advertising. The valid range is 0x00 - 0x03. |
| [in] | duration | The duration for which the advertising set identified by adv_hdl is enabled. Time = duration * 10ms. When the duration expires, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The valid range is 0x0000 - 0xFFFF. The duration parameter is ignored when the value is set to 0x0000. |
| [in] | max_extd_adv_evts | The maximum number of advertising events that be sent during advertising. When all the advertising events(max_extd_adv_evts) have been sent, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The max_extd_adv_evts parameter is ignored when the value is set to 0x00. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | adv_hdl is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StopAdv()

ble_status_t R_BLE_GAP_StopAdv ( uint8_t *adv_hdl*)

Stop advertising.

This function stops advertising. The result of this API call is notified in BLE_GAP_EVENT_ADV_OFF event.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | The advertising handle pointing to the advertising set which stops advertising. The valid range is 0x00 - 0x03. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | adv_hdl is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetPerdAdvParam()

ble_status_t R_BLE_GAP_SetPerdAdvParam ( st_ble_gap_perd_adv_param_t * *p_perd_adv_param*)

Set periodic advertising parameters.

This function sets periodic advertising parameters. Create the advertising set which supports Non-Connectable, Non-Scannable advertising by R_BLE_GAP_SetAdvParam() before setting periodic advertising parameters. The result of this API call is notified in BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_perd_adv_param | Periodic advertising parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_perd_adv_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The following field in the p_perd_adv_param parameter is out of range.<br>• adv_hdl<br>• perd_intv_min or perd_intv_max<br>• prop_type is neither 0x0000 nor 0x0040(BLE_GAP_PERD_PROP_TX_POWER) |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StartPerdAdv()

ble_status_t R_BLE_GAP_StartPerdAdv ( uint8_t *adv_hdl*)

Start periodic advertising.

This function starts periodic advertising. Set periodic advertising parameters to the advertising set, before starting periodic advertising. The result of this API call is notified in BLE_GAP_EVENT_PERD_ADV_ON event.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | Advertising handle identifying the advertising set which starts periodic advertising. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | adv_hdl is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StopPerdAdv()

ble_status_t R_BLE_GAP_StopPerdAdv ( uint8_t *adv_hdl*)

Stop periodic advertising.

This function stops periodic advertising. If the return value of this API is BLE_SUCCESS, the result is notified in BLE_GAP_EVENT_PERD_ADV_OFF event.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | Specify the handle of Advertising Set to stop Periodic Advertising. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | adv_hdl is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_GetRemainAdvBufSize()

ble_status_t R_BLE_GAP_GetRemainAdvBufSize ( uint16_t * *p_remain_adv_data_size*, uint16_t * *p_remain_perd_adv_data_size* )

Get buffer size for advertising data/scan response data/periodic advertising data in the Controller.

This function gets the total size of advertising data/scan response data/periodic advertising data which can be currently set to Controller(all of the advertising sets). The application layer gets the data sizes via the parameters. By this API function call, no events occur.

**Parameters**

| | | |
|---|---|---|
| [out] | p_remain_adv_data_size | The free buffer size of Controller to which advertising data/scan response data can be currently set. |
| [out] | p_remain_perd_adv_data_size | The free buffer size of Controller to which periodic advertising data can be currently set. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_remain_adv_data_size or p_remain_perd_adv_data_size is specified as NULL. |

### ◆ R_BLE_GAP_RemoveAdvSet()

ble_status_t R_BLE_GAP_RemoveAdvSet ( uint8_t *op_code*, uint8_t *adv_hdl* )

Delete advertising set.

This function deletes an advertising set or deletes all the advertising sets. The result of this API call is notified in BLE_GAP_EVENT_ADV_SET_REMOVE_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | op_code | The operation for delete or clear.<br>macro: BLE_GAP_RMV_ADV_SET_REM_OP(0x01) — Delete an advertising set.<br>BLE_GAP_RMV_ADV_SET_CLR_OP(0x02) — Delete all the advertising sets. |
| [in] | adv_hdl | Advertising handle identifying the advertising set deleted. If op_code is BLE_GAP_RMV_ADV_SET_CLR_OP, adv_hdl is ignored. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• op_code is out of range.<br>• When op_code is BLE_GAP_RMV_ADV_SET_REM_OP(0x01), adv_hdl is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_CreateConn()

ble_status_t R_BLE_GAP_CreateConn ( st_ble_gap_create_conn_param_t * *p_param*)

Request for a link establishment.

This function sends a connection request to a remote device to create a link. When Controller has received a request for establishment of a link from host stack, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified to the application layer. When the link is established, BLE_GAP_EVENT_CONN_IND event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_param | Connection parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• p_param is specified as NULL.<br>• p_conn_param_1M field and p_conn_param_2M and p_conn_param_coded field in p_param are specified as NULL.<br>• When creating a link with 1M PHY, p_conn_param in p_conn_param_1M field in p_param is specified as NULL.<br>• When creating a link with 2M PHY, p_conn_param in p_conn_param_2M field in p_param is specified as NULL.<br>• When creating a link with coded MPHY, p_conn_param in p_conn_param_coded field in p_param is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• init_filter_policy in p_param is out of range.<br>• remote_bd_addr_type field or own_addr_type address field in p_param is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_CancelCreateConn()

| ble_status_t R_BLE_GAP_CancelCreateConn ( void ) |
|---|

Cancel the request for a link establishment.

This function cancels a request for establishing a link. When Controller has received the cancel request from host stack, BLE_GAP_EVENT_CONN_CANCEL_COMP event is notified to the application layer. When the cancel procedure has completed, BLE_GAP_EVENT_CONN_IND event is notified to the application layer.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetChMap()

| ble_status_t R_BLE_GAP_SetChMap ( uint8_t * *p_channel_map*) |
|---|

Set the Channel Map.

This function sets the channel map. The result of this API call is notified in BLE_GAP_EVENT_CH_MAP_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_channel_map | Channel map. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_channel_map is specified as NULL. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StartScan()

| ble_status_t R_BLE_GAP_StartScan ( st_ble_gap_scan_param_t * *p_scan_param*, st_ble_gap_scan_on_t * *p_scan_enable* ) |
|---|

Set scan parameter and start scan.

This function starts scanning. When scanning for the first time, set the p_scan_param. Setting scan parameters can be omitted by specifying p_scan_param as NULL after next time. The result of this API call is notified in BLE_GAP_EVENT_SCAN_ON event. Advertising report is notified in BLE_GAP_EVENT_ADV_REPT_IND event. Figure 1.3 shows the relationship between scan period, scan duration, scan interval and scan window.

Figure 351: Figure 1.3

When scan duration is non-zero, scan period is zero and scan duration expires, BLE_GAP_EVENT_SCAN_TO event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_scan_param | Scan parameter. When p_scan_param is specified as NULL, host stack doesn't set scan parameters and start scanning with the previous parameters. |
| [in] | p_scan_enable | Scan period, scan duration, duplicate filter and procedure type. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• p_scan_enable is specified as NULL.<br>• p_phy_param_1M field and p_phy_param_coded field in p_scan_param are specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• proc_type field in p_scan_enable is out of range.<br>• filter_dups in p_scan_enable is out of range.<br>• o_addr_type in p_scan_param is out of range.<br>• filter_policy in p_scan_param is out of range.<br>• scan_type of p_scan_param's p_phy_param_1M or p_phy_param_coded is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_StopScan()

ble_status_t R_BLE_GAP_StopScan ( void )

Stop scan.

This function stops scanning. The result of this API call is notified in BLE_GAP_EVENT_SCAN_OFF event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_CreateSync()

ble_status_t R_BLE_GAP_CreateSync ( st_ble_dev_addr_t * *p_addr*, uint8_t *adv_sid*, uint16_t *skip*, uint16_t *sync_to* )

Request for a periodic sync establishment.

This function sends a request for establishment of a periodic sync to a advertiser. In order to create a periodic sync, scan needs to be starting by R_BLE_GAP_StartScan(). When Controller has received the request from host stack, BLE_GAP_EVENT_CREATE_SYNC_COMP event is notified to the application layer. When the periodic sync is established, BLE_GAP_EVENT_SYNC_EST event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_addr | The address of periodic advertiser.When p_addr is specified as NULL, local device creates a periodic sync with the advertiser registered in Periodic Advertiser List. |
| [in] | adv_sid | Advertising SID. When p_addr is specified as NULL, adv_sid is ignored. Valid range is 0x00 - 0x0F. |
| [in] | skip | The number of consecutive periodic advertising packets that local device may skip after receiving a periodic advertising packet. Valid range is 0x0000 - 0x01F3. |
| | | |

| | | |
|---|---|---|
| [in] | sync_to | The maximum permitted time between successful receives.When sync_to expires, the periodic sync is lost. Time(ms) = sync_to * 10. Valid range is 0x000A - 0x4000. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_addr is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The following parameter is out of range.<br>• address type in p_addr<br>• adv_sid<br>• skip<br>• sync_to |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_CancelCreateSync()

ble_status_t R_BLE_GAP_CancelCreateSync ( void )

Cancel the request for a periodic sync establishment.

This function cancels a request for establishing a periodic sync. The result of this API call is notified in BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_TerminateSync()

ble_status_t R_BLE_GAP_TerminateSync ( uint16_t *sync_hdl*)

Terminate the periodic sync.

This function terminates a periodic sync. The result of this API call is notified in BLE_GAP_EVENT_SYNC_TERM event.

**Parameters**

| | | |
|---|---|---|
| [in] | sync_hdl | Sync handle identifying the periodic sync to be terminated. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | sync_hdl is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_ConfPerdAdvList()

ble_status_t R_BLE_GAP_ConfPerdAdvList ( uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, uint8_t * *p_adv_sid_set*, uint8_t *device_num* )

Set Periodic Advertiser List.

This function supports the following operations regarding Periodic Advertiser List.

- Add the device to Periodic Advertiser List.
- Delete the device from Periodic Advertiser List.

- Clear Periodic Advertiser List.

  The total number of Periodic Advertiser List entries is defined as BLE_GAP_PERD_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_PERD_LIST_CONF_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | op_code | The operation for Periodic Advertiser List.<br>macro: BLE_GAP_LIST_ADD_DEV(0x01) — Add the device to the list.<br>BLE_GAP_LIST_REM_DEV(0x02) — Delete the device from the list.<br>BLE_GAP_LIST_CLR(0x03) — Clear the list. |
| [in] | p_addr | An array of device address to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored. |
| [in] | p_adv_sid_set | An array of SID of the advertiser to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_adv_sid_set is ignored. |
| [in] | device_num | The number of devices add / delete to the list. |

| | | |
|---|---|---|
| | | Valid range is 1-BLE_GAP_PERD_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, p_addr or p_adv_sid_set is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | op_code or address type field in p_addr or p_adv_sid_set or device_num is out of range. |
| BLE_ERR_UNSUPPORTED(0x0007) | Not supported. |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• While operating Periodic Advertiser List, this function was called.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for operating periodic advertiser. |

### ◆ R_BLE_GAP_AuthorizeDev()

ble_status_t R_BLE_GAP_AuthorizeDev ( uint16_t *conn_hdl*, uint8_t *author_flag* )

Authorize a remote device.

User authorizes a remote device by this function. This function is used when a remote device accesses a GATT Characteristic in local device which requests user authorization. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to be authorized or not by user. |
| [in] | author_flag | Authorize or not the remote device.<br>macro: description<br>BLE_GAP_NOT_AUTHORIZED(0x00): Not authorize the remote device.<br>BLE_GAP_AUTHORIZED(0x01): Authorize the remote device. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | author_flag is out of range. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_GetRemDevInfo()

ble_status_t R_BLE_GAP_GetRemDevInfo ( uint16_t *conn_hdl*)

Get the information about remote device.

This function retrieves information about the remote device. The information includes BD_ADDR, the version number and LE features. The result of this API call is notified in BLE_GAP_EVENT_GET_REM_DEV_INFO event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device whose information to be retrieved. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetPairingParams()

ble_status_t R_BLE_GAP_SetPairingParams ( st_ble_gap_pairing_param_t * *p_pair_param*)

Set the parameters using pairing.

This function sets the parameters used in pairing. The parameters set by this API are sent to the remote device when pairing occurred. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_pair_param | Pairing parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The following field in p_pair_param is out of range.<br>• iocap<br>• max_key_size<br>• mitm<br>• bonding<br>• key_notf<br>• sec_conn_only |

### ◆ R_BLE_GAP_SetLocIdInfo()

ble_status_t R_BLE_GAP_SetLocIdInfo ( st_ble_dev_addr_t * *p_lc_id_addr*, uint8_t * *p_lc_irk* )

Set the IRK and the identity address distributed to a remote device.

This function registers local IRK and identity address of local device in host stack. The IRK and the identity address are distributed to a remote device in pairing. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_lc_id_addr | Identity address to be registered in host stack. |
| [in] | p_lc_irk | IRK to be registered in host stack. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_lc_id_addr or p_lc_irk is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | Address type field in p_lc_id_addr is out of range. |

### ◆ R_BLE_GAP_SetLocCsrk()

ble_status_t R_BLE_GAP_SetLocCsrk ( uint8_t * *p_local_csrk*)

Set the CSRK distributed to a remote device.

This function registers local CSRK in host stack. The CSRK is distributed to a remote device in pairing. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_local_csrk | CSRK to be registered in host stack. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_local_csrk is specified as NULL. |

### ◆ R_BLE_GAP_StartPairing()

| ble_status_t R_BLE_GAP_StartPairing ( uint16_t *conn_hdl*) |
|---|

Start pairing.

This function starts pairing with a remote device. The result of this API call is returned by a return value. The result of pairing is notified in BLE_GAP_EVENT_PAIRING_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device which local device starts pairing with. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | While generating OOB data, this function was called. |
| BLE_ERR_CONTEXT_FULL(0x000B) | While pairing, this function was called. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_ReplyPairing()

ble_status_t R_BLE_GAP_ReplyPairing ( uint16_t *conn_hdl*, uint8_t *response* )

Reply the pairing request from a remote device.

This function replies to the pairing request from the remote device. The pairing request from the remote device is notified in BLE_GAP_EVENT_PAIRING_REQ event. The result of this API call is returned by a return value. The result of pairing is notified in BLE_GAP_EVENT_PAIRING_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device which local device starts pairing with. |
| [in] | response | Accept or reject the pairing request from the remote device.<br>macro / description<br>BLE_GAP_PAIRING_ACCEPT(0x00) — Accept the pairing request.<br>BLE_GAP_PAIRING_REJECT(0x01) — Reject the pairing request. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | response is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | While generating OOB data, this function was called. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | When this function was called, host stack has not yet received BLE_GAP_EVENT_PAIRING_REQ event. |

### ◆ R_BLE_GAP_StartEnc()

ble_status_t R_BLE_GAP_StartEnc ( uint16_t *conn_hdl*)

Encryption the link.

This function starts encryption of the link. In case of master device, the local device requests for the encryption to a remote device. In case of slave device, the local device sends a Security Request to a remote device. After receiving the Security Request, the remote device requests for the encryption to the local device. The result of the encryption is returned in BLE_GAP_EVENT_ENC_CHG event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link which is encrypted. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• Pairing has not been completed.<br>• The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_ReplyPasskeyEntry()

ble_status_t R_BLE_GAP_ReplyPasskeyEntry ( uint16_t *conn_hdl*, uint32_t *passkey*, uint8_t *response* )

Reply the passkey entry request.

When BLE_GAP_EVENT_PASSKEY_ENTRY_REQ event is notified, the response to passkey entry is sent by this function. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device which the reply to passkey entry is sent. |
| [in] | passkey | Passkey. The valid range is 000000 - 999999 in decimal. |
| [in] | response | Active or negative reply to passkey entry.<br>macro / description<br>BLE_GAP_PAIRING_ACCEPT(0x00) — Accept the passkey entry pairing.<br>BLE_GAP_PAIRING_REJECT(0x01) — Reject the passkey entry pairing. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | passkey or response is out of range. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | When this function was called, pairing has not yet started. |

### ◆ R_BLE_GAP_ReplyNumComp()

ble_status_t R_BLE_GAP_ReplyNumComp ( uint16_t *conn_hdl*, uint8_t *response* )

Reply the numeric comparison request.

When BLE_GAP_EVENT_NUM_COMP_REQ event is notified, the response to Numeric Comparison is sent by this function. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device which the reply to Numeric Comparison is sent. |
| [in] | response | Active or negative reply in Numeric Comparison.<br>macro: BLE_GAP_PAIRING_ACCEPT(0x00) — The number displayed in the local is the same as the one of the remote.<br>macro: BLE_GAP_PAIRING_REJECT(0x01) — The number displayed in the local is differs from the one of the remote. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | response is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | When this function was called, host stack has not yet received BLE_GAP_EVENT_NUM_COMP_REQ event. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | When this function was called, pairing has not yet started. |

### ◆ R_BLE_GAP_NotifyKeyPress()

ble_status_t R_BLE_GAP_NotifyKeyPress ( uint16_t *conn_hdl*, uint8_t *key_press* )

Notify the input key type which a remote device inputs in the passkey entry.

This function notifies the input key type to the remote device in passkey entry. The result is returned from this API.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to which the key notification is sent. |
| [in] | key_press | Input key type.<br>macro / description<br>BLE_GAP_LESC_PASSKEY_ENTRY_STARTED(0x00) — Notify that passkey entry started.<br>BLE_GAP_LESC_PASSKEY_DIGIT_ENTERED(0x01) — Notify that passkey digit entered.<br>BLE_GAP_LESC_PASSKEY_DIGIT_ERASED(0x02) — Notify that passkey digit erased.<br>BLE_GAP_LESC_PASSKEY_CLEARED(0x03) — Notify that passkey cleared.<br>BLE_GAP_LESC_PASSKEY_ENTRY_COMPLETED(0x04) — Notify that passkey entry completed. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | key_press parameter is out of range. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | When this function was called, pairing has not yet started. |

### ◆ R_BLE_GAP_GetDevSecInfo()

ble_status_t R_BLE_GAP_GetDevSecInfo ( uint16_t *conn_hdl*, st_ble_gap_auth_info_t * *p_sec_info* )

Get the security information about the remote device.

This function gets the parameters which has been negotiated with the remote device in pairing. The parameters can be retrieved after pairing. The result is returned by p_sec_info.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device whose bonding information is retrieved. |
| [in] | p_sec_info | Return the security information which has been negotiated in pairing. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_sec_info is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The remote device bonding information has not been set to host stack. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_ReplyExKeyInfoReq()

ble_status_t R_BLE_GAP_ReplyExKeyInfoReq ( uint16_t *conn_hdl*)

Distribute the keys of local device.

When key exchange request is notified by BLE_GAP_EVENT_EX_KEY_REQ event at pairing, keys of the local device are distributed. The result is returned from this API.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to which the key is distributed. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | When this function was called, pairing has not yet started. |

### ◆ R_BLE_GAP_SetRemOobData()

ble_status_t R_BLE_GAP_SetRemOobData ( st_ble_dev_addr_t * *p_addr*, uint8_t *oob_data_flag*, st_ble_gap_oob_data_t * *p_oob* )

Set the oob data from a remote device.

This function registers the OOB data received from a remote device. When oob_data_flag indicates that the OOB data has been received, the setting regarding OOB data is reflected in pairing. In order to do OOB pairing, set the OOB data received from the remote device before pairing. The result is returned from this API.

**Parameters**

| | | |
|---|---|---|
| [in] | p_addr | The remote device address. |
| [in] | oob_data_flag | This parameter indicates whether the local device has received the OOB data from the remote device or not.<br>macro: BLE_GAP_OOB_DATA_NOT_PRESENT(0x00) — Reply that No OOB data has been received when pairing.<br>macro: BLE_GAP_OOB_DATA_PRESENT(0x01) — Reply that the OOB data has been received when pairing. |
| [in] | p_oob | The OOB data received from the remote device. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• p_addr is specified as NULL.<br>• oob_data_flag is BLE_GAP_OOB_DATA_PRESENT and p_oob is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | oob_data_flag is out of range. |
| BLE_ERR_CONTEXT_FULL(0x000B) | There is no room to register the OOB data received from a remote device. |

### ◆ R_BLE_GAP_CreateScOobData()

| ble_status_t R_BLE_GAP_CreateScOobData ( void ) |
|---|

Create data for oob in secure connection.

This function generates the OOB data distributed to a remote device in Secure Connections. The result of this API call is notified in BLE_GAP_EVENT_SC_OOB_CREATE_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The reason for this error is as follows:<br>• This function was called in pairing.<br>• The task for host stack is not running. |
| BLE_ERR_ALREADY_IN_PROGRESS(0x000A) | This function was called in creating OOB data. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |

### ◆ R_BLE_GAP_SetBondInfo()

ble_status_t R_BLE_GAP_SetBondInfo ( st_ble_gap_bond_info_t * *p_bond_info*, uint8_t *device_num*, uint8_t * *p_set_num* )

Set the bonding information stored in non-volatile memory to the host stack.

Set the bonding information of the remote device in the host stack. After power re-supply, when the remote device bonding information stored in non-volatile memory is set to host stack, this function is used. Host stack can be set the number specified by the device_num parameter of bonding information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_bond_info | An array of bonding information. The number of elements is specified by device_num. |
| [in] | device_num | The number of the devices of which host stack registers bonding information. |
| [in] | p_set_num | The number of the devices whose bonding information was registered in host stack. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_bond_info or p_set_num is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | device_num is out of range. |
| BLE_ERR_CONTEXT_FULL(0x000B) | Host stack already has the maximum number of bonding information. |

### ◆ R_BLE_GAP_DeleteBondInfo()

void R_BLE_GAP_DeleteBondInfo ( int32_t *local*, int32_t *remote*, st_ble_dev_addr_t * *p_addr*, ble_gap_del_bond_cb_t *gap_del_bond_cb* )

This function deletes the bonding information in Host Stack.
When a function for deleting the bonding information stored in non-volatile area is registered by the gap_del_bond_cb parameter, it is deleted as well as the bonding information in Host Stack.

**Parameters**

| | | |
|---|---|---|
| [in] | local | The type of the local bonding information to be deleted.<br>macro: description<br>BLE_GAP_SEC_DEL_LOC_NONE(0x00): Delete no local keys.<br>BLE_GAP_SEC_DEL_LOC_IRK(0x01): Delete local IRK and identity address.<br>BLE_GAP_SEC_DEL_LOC_CSRK(0x02): Delete local CSRK.<br>BLE_GAP_SEC_DEL_LOC_ALL(0x03): Delete all local keys. |
| [in] | remote | The type of the remote bonding information to be deleted.<br>macro: description<br>BLE_GAP_SEC_DEL_REM_NONE(0x00): Delete no remote device keys.<br>BLE_GAP_SEC_DEL_REM_SA(0x01): Delete the keys specified by the p_addr parameter.<br>BLE_GAP_SEC_DEL_REM_NOT_CONN(0x02): Delete keys of not connected remote devices.<br>BLE_GAP_SEC_DEL_REM_ALL(0x03): Delete all remote device keys. |

| | | |
|---|---|---|
| | | |
| [in] | p_addr | p_addr is specified as the address of the remote device whose keys are deleted when the rem_info parameter is set to BLE_GAP_SEC_DEL_REM_SA(0x01). |
| [in] | gap_del_bond_cb | This parameter is a callback function which deletes the bonding information stored in non-volatile area.<br>After deleting the bonding information stored in Host Stack, the callback function is called. If no bonding information is stored in non-volatile area, specify the parameter as NULL. |

**Return values**

| | |
|---|---|
| none | |

### ◆ R_BLE_GAP_ReplyLtkReq()

ble_status_t R_BLE_GAP_ReplyLtkReq ( uint16_t *conn_hdl*, uint16_t *ediv*, uint8_t * *p_peer_rand*, uint8_t *response* )

Reply the LTK request from a remote device.

This function replies to the LTK request in BLE_GAP_EVENT_LTK_REQ event from a remote device. The result of the LTK reply is returned in BLE_GAP_EVENT_LTK_RSP_COMP event. When the link encryption has completed, BLE_GAP_EVENT_ENC_CHG event is notified.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device which sent the LTK request. |
| [in] | ediv | Ediv notified in BLE_GAP_EVENT_LTK_REQ event. |
| [in] | p_peer_rand | Rand notified in BLE_GAP_EVENT_LTK_REQ event. |
| [in] | response | Response to the LTK request. If |

| | | |
|---|---|---|
| | | "BLE_GAP_LTK_REQ_ACCEPT " is specified, when no LTK has been exchanged in pairing, reject the LTK request.<br>macro / description<br>BLE_GAP_LTK_REQ_ACCEPT(0x00) — Reply for the LTK request.<br>BLE_GAP_LTK_REQ_DENY(0x01) — Reject the LTK request. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | p_peer_rand is specified as NULL in case of legacy pairing. |
| BLE_ERR_INVALID_ARG(0x0003) | response is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |

### ◆ R_BLE_GAP_SetCteConnlessParam()

ble_status_t R_BLE_GAP_SetCteConnlessParam ( st_ble_gap_cte_connless_t * *p_cte_param*)

Set the parameters for the transmission of Constant Tone Extensions in any periodic advertising.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cte_param | parameters of type, length, and antenna switching pattern. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_EnableCteConnless()

ble_status_t R_BLE_GAP_EnableCteConnless ( uint16_t *adv_hdl*, uint8_t *enable* )

Enable or disable Constant Tone Extensions in periodic advertising identified by the adv_hdl.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | handle of the periodic advertising which carries the CTE info. |
| [in] | enable | Enable or disable address resolution function.<br>macro: BLE_GAP_CTE_DISABLED (0x00) — Disable connectionless CTE transmission<br>BLE_GAP_CTE_ENABLED( 0x01) — Enable connectionless CTE transmission |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StartCteConnlessRecv()

ble_status_t R_BLE_GAP_StartCteConnlessRecv ( st_ble_gap_cte_connless_recv_t * *p_cte_recv*)

Enable sampling received Constant Tone Extension fields.

Application should receive BLE_GAP_EVENT_CTE_CONNLESS_REPT event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cte_recv | antenna switching pattern and switching and sampling slot durations to be used. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StopCteConnlessRecv()

ble_status_t R_BLE_GAP_StopCteConnlessRecv ( uint16_t *sync_hdl*)

Disable sampling received Constant Tone Extension fields.

**Parameters**

| | | |
|---|---|---|
| [in] | sync_hdl | handle of the periodic advertising sync which carries the CTE info. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetCteConnParam()

ble_status_t R_BLE_GAP_SetCteConnParam ( st_ble_gap_cte_conn_t * *p_cte_param*)

Set the parameters for the transmission of Constant Tone Extensions in ACL link.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cte_param | parameters of type, length, and antenna switching pattern. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_EnableCteConnRsp()

ble_status_t R_BLE_GAP_EnableCteConnRsp ( uint16_t *conn_hdl*, uint8_t *enable* )

Enable or disable Constant Tone Extensions Transmission in ACL link by conn_hdl.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | handle of the ACL link which carries the CTE info. |
| [in] | enable | Enable or disable address resolution function.<br>macro: BLE_GAP_CTE_DISABLED (0x00) — Disable connectionless CTE transmission<br>BLE_GAP_CTE_ENABLED( 0x01) — Enable connectionless CTE transmission |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetCteConnRecvParam()

ble_status_t R_BLE_GAP_SetCteConnRecvParam ( st_ble_gap_cte_conn_rx_param_t * *p_cte_param*)

Set the parameters for the receiving of Constant Tone Extensions in ACL link. and start sampling.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cte_param | parameters of type, length, and antenna switching pattern. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StopCteConnRecvSampling()

ble_status_t R_BLE_GAP_StopCteConnRecvSampling ( uint16_t *conn_hdl*)

Stop sampling of Constant Tone Extensions on the specified connection.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | handle of the ACL link which carries the CTE info. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StartCteConnReq()

ble_status_t R_BLE_GAP_StartCteConnReq ( st_ble_gap_cte_conn_req_t * *p_req*)

Set the parameters and start sending request of Constant Tone Extensions in ACL link to peer.

If the request does not receive a CTE Response PDU with CTE info, BLE_GAP_EVENT_CTE_CONN_REQ_FAILED event is sent to application. Otherwise application should receive BLE_GAP_EVENT_CTE_CONN_REPT event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_req | parameters connection CTE request. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StopCteConnReq()

ble_status_t R_BLE_GAP_StopCteConnReq ( uint16_t *handle*)

Stop sending request of Constant Tone Extensions in ACL link to peer.

**Parameters**

| | | |
|---|---|---|
| [in] | handle | handle of the ACL link which carries the CTE info. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetDefaultSubrate()

ble_status_t R_BLE_GAP_SetDefaultSubrate ( st_ble_gap_subrate_param_t * *p_subrate_param*)

Set the initial values for the acceptable parameters for subrating requests,.

**Parameters**

| | | |
|---|---|---|
| [in] | p_subrate_param | default parameters of subrate. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_RequestSubrate()

ble_status_t R_BLE_GAP_RequestSubrate ( uint16_t *conn_hdl*, st_ble_gap_subrate_param_t * *p_subrate_param* )

Request a change to the subrating factor other parameters applied to an existing connection using the Connection Subrate Update procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | handle of the ACL link which carries the CTE info. |
| [in] | p_subrate_param | request parameters of subrate. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StartPerdAdvSetInfoTransfer()

ble_status_t R_BLE_GAP_StartPerdAdvSetInfoTransfer ( uint16_t *adv_hdl*, uint16_t *conn_hdl*, uint16_t *service_data* )

This function starts Periodic advertising adv set info transfer to the connection.

Send synchronization information about the periodic advertising in an advertising set to a connected device.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | Identifies the advertising set. |
| [in] | conn_hdl | Connection handle identifying the remote device. |
| [in] | service_data | A value provided by the application for use by the application of the peer device. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_StartPerdAdvSyncTransfer()

ble_status_t R_BLE_GAP_StartPerdAdvSyncTransfer ( uint16_t *sync_hdl*, uint16_t *conn_hdl*, uint16_t *service_data* )

This function starts Periodic advertising sync transfer.

send synchronization information about the periodic advertising train identified by the Sync_Handle parameter to a connected device.

**Parameters**

| | | |
|---|---|---|
| [in] | sync_hdl | Sync handle identifying the Periodic Sync that has been established. |
| [in] | conn_hdl | Connection handle identifying the remote device. |
| [in] | service_data | A value provided by the application for use by the application of the peer device. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetPerdAdvSyncTransferParam()

ble_status_t R_BLE_GAP_SetPerdAdvSyncTransferParam ( uint16_t *conn_hdl*, st_ble_gap_past_param_t * *p_past_param* )

This function starts to accept Periodic advertising sync transfer from the connection.

This API call enables BLE_GAP_EVENT_PAST_RECV event.n

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device. |
| [in] | p_past_param | Periodic advertising sync transfer parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetDefPerdAdvSyncTransferParam()

ble_status_t R_BLE_GAP_SetDefPerdAdvSyncTransferParam ( st_ble_gap_past_param_t * *p_past_param*)

This function set the default parameter of Periodic advertising sync transfer for all subsequent connection. It does not affect any existing connection.

This API call enables BLE_GAP_EVENT_PAST_RECV event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_past_param | Periodic advertising sync transfer parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_ReadAntennaInfo()

ble_status_t R_BLE_GAP_ReadAntennaInfo ( void )

This function read the switching rates, the sampling rates, the number of antennae, and the maximum length of a transmitted Constant Tone Extension.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |
| BLE_ERR_UNSPECIFIED(0x0013) | Unspecified error. |

### ◆ R_BLE_GAP_ReceiverTest()

ble_status_t R_BLE_GAP_ReceiverTest ( st_ble_gap_recv_test_param_t * *p_rx_test_param*)

Start a test where the DUT receives test reference packets at a fixed interval. The tester generates the test reference packets.

**Parameters**

| | | |
|---|---|---|
| [in] | p_rx_test_param | receiver test parameter |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_TransmitterTest()

ble_status_t R_BLE_GAP_TransmitterTest ( st_ble_gap_trans_test_param_t * *p_tx_test_param*)

Start a test where the DUT generates test reference packets at a fixed interval. The Controller shall transmit at the power level indicated by the TX_Power_Level parameter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_tx_test_param | transmitter test parameter |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_ModifySleepClockAccuracy()

ble_status_t R_BLE_GAP_ModifySleepClockAccuracy ( uint8_t *act*)

request that the Controller changes its sleep clock accuracy for testing purposes. It should not be used under other circumstances.

**Parameters**

| | | |
|---|---|---|
| [in] | act | Switch to a more/less accurate clock. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_ReadRemoteTransmitPowerLevel()

ble_status_t R_BLE_GAP_ReadRemoteTransmitPowerLevel ( uint16_t *conn_hdl*, uint8_t *phy* )

Read the transmit power level used by the remote device.

BLE_GAP_EVENT_TX_POWER_REPT is received as a result when R_BLE_GAP_SetTransmitPowerReportingEnable is enabled.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle. |
| [in] | phy | The transmitter PHY of packets. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_GAP_SetPathLossReportingParam()

ble_status_t R_BLE_GAP_SetPathLossReportingParam ( st_ble_gap_set_path_loss_rpt_param_t * *p_loss_rpt_param*)

Set the path loss threshold reporting parameters for the ACL connection identified.

**Parameters**

| | | |
|---|---|---|
| [in] | p_loss_rpt_param | parameter for path loss report. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_SetPathLossReportingEnable()

ble_status_t R_BLE_GAP_SetPathLossReportingEnable ( uint16_t *conn_hdl*, uint8_t *enable* )

Enable or disable path loss reporting for the ACL connection.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle. |
| [in] | enable | Reporting enable/disable. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_SetTransmitPowerReportingEnable()

ble_status_t R_BLE_GAP_SetTransmitPowerReportingEnable ( uint16_t *conn_hdl*, uint8_t *local_enable*, uint8_t *remote_enable* )

Enable or disable the transmit power level changing report.

Enable or disable the reporting to the local Host of transmit power level changes in the local and remote Controllers for the ACL connection identified by the Connection_Handle parameter.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle. |
| [in] | local_enable | Local transmit power reports enable/disable. |
| [in] | remote_enable | Remote transmit power reports enable/disable. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_SetDataRelatedAddrChanges()

ble_status_t R_BLE_GAP_SetDataRelatedAddrChanges ( uint8_t *adv_hdl*, uint8_t *change_reason* )

Specifies circumstances when the Controller shall refresh any Resolvable Private Address.

Specifies circumstances when the Controller shall refresh any Resolvable Private Addresss used by the advertising set identified by the Advertising_Handle parameter, whether or not the address timeout period has been reached. This function may be used while advertising is enabled.

**Parameters**

| | | |
|---|---|---|
| [in] | adv_hdl | Used to identify an advertising set. |
| [in] | change_reason | reason(s) for refreshing addresses. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_TestEnd()

ble_status_t R_BLE_GAP_TestEnd ( void )

Stop any test which is in progress. The Num_Packets for a transmitter test shall be reported as 0x0000. The Num_Packets is an unsigned number and contains the number of received packets.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_ReqPeerSCA()

ble_status_t R_BLE_GAP_ReqPeerSCA ( uint16_t *conn_hdl*)

Read the Sleep Clock Accuracy (SCA) of the peer device.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Indicate an ACL connection. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_EnhancedReadTxPowerLevel()

ble_status_t R_BLE_GAP_EnhancedReadTxPowerLevel ( uint16_t *conn_hdl*, uint8_t *phy* )

Read the current and maximum transmit power levels of the local Controller on the ACL connection identified by the Connection_Handle parameter and the PHY indicated by the PHY parameter.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Indicate an ACL connection. |
| [in] | phy | Indicate PHY. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_GAP_SetHostFeat()

ble_status_t R_BLE_GAP_SetHostFeat ( uint8_t *bit_number*, uint8_t *bit_value* )

Set or clear a bit controlled by the Host in the Link Layer FeatureSet stored in the Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | bit_number | Bit position in the FeatureSet |
| [in] | bit_value | Value of the bit |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

## ISO

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_ISO_CreateBig (uint8_t *p_big_hdl, uint8_t adv_hdl, st_ble_iso_big_param_t *p_big_param)<br>Create a BIG. More... |
| ble_status_t | R_BLE_ISO_StopBig (uint8_t big_hdl, uint8_t reason)<br>Stop a BIG of the big_handle. More... |

| | |
|---|---|
| ble_status_t | R_BLE_ISO_CreateBigSync (uint8_t *p_big_hdl, uint16_t sync_hdl, st_ble_iso_big_sync_param_t *p_big_sync_param)<br>Create a BIG sync. More... |
| ble_status_t | R_BLE_ISO_TerminateBigSync (uint8_t big_hdl)<br>Terminate a BIG sync. More... |
| ble_status_t | R_BLE_ISO_SetCigParam (uint8_t *p_cig_id, st_ble_iso_cig_param_t *p_cig_param)<br>Create a CIG with param. More... |
| ble_status_t | R_BLE_ISO_CreateCis (st_ble_iso_cis_conn_t *p_cis_conn)<br>Create one or more CISes using the CIS param. More... |
| ble_status_t | R_BLE_ISO_RemoveCig (uint8_t cig_id)<br>remove a CIG of id and all of CIS streams in this CIG. More... |
| ble_status_t | R_BLE_ISO_ReplyCisRequest (uint8_t cig_id, uint8_t cis_id, uint8_t response, uint8_t reason)<br>Reply the CIS request from a remote device. More... |
| ble_status_t | R_BLE_ISO_SetupDataPath (uint16_t conn_hdl, st_ble_iso_chan_path *p_path)<br>Create the ISO data path between the Host and the Controller for a CIS. More... |
| ble_status_t | R_BLE_ISO_SendData (st_ble_iso_sdu_t *p_sdu_info)<br>Send a SDU payload to a ISO channel of conn_hdl. More... |
| ble_status_t | R_BLE_ISO_SendDataNoCopy (st_ble_iso_sdu_t *p_sdu_info)<br>Send SDU payload to a ISO channel of conn_hdl without copying data. More... |
| ble_status_t | R_BLE_ISO_CreateBigTest (uint8_t *p_big_hdl, uint8_t adv_hdl, st_ble_iso_create_big_test_param_t *p_create_big_test_param)<br>Create one or more BISes of a BIG (see [Vol 6] Part B, Section 4.4.6). |

| | |
|---|---|
| | All BISes in the BIG have the same values for all parameters. More... |
| ble_status_t | R_BLE_ISO_SetCigParamTest (uint8_t *p_cig_id, st_ble_iso_set_cig_param_test_param_t *p_set_cig_param_test_param)<br>Create a CIG and set the parameters of one or more CISes that are associated with a CIG in the Controller. More... |
| ble_status_t | R_BLE_ISO_TransmitTest (uint16_t conn_hdl, uint8_t payload_type)<br>Configure an established CIS or BIS and transmit test payloads which are generated by the Controller. More... |
| ble_status_t | R_BLE_ISO_ReceiveTest (uint16_t conn_hdl, uint8_t payload_type)<br>Configure an established CIS or a synchronized BIG to receive payloads. More... |
| ble_status_t | R_BLE_ISO_ReadTestCounters (uint16_t conn_hdl)<br>Read the test counters (see [Vol 6] Part B, Section 7) in the Controller which is configured in ISO Receive Test mode for a CIS or BIS specified by the Connection_Handle. Reading the test counters does not reset the test counters. More... |
| ble_status_t | R_BLE_ISO_TestEnd (uint16_t conn_hdl)<br>Terminate the ISO Transmit and/or Receive Test mode for a CIS or BIS specified by the Connection_Handle parameter but does not terminate the CIS or BIS. More... |
| ble_status_t | R_BLE_ISO_ReadLinkQuality (uint16_t conn_hdl)<br>Returns the values of various counters related to link quality that are associated with the isochronous stream specified by the Connection_Handle parameter. More... |
| ble_status_t | R_BLE_ISO_RemoveDataPath (uint16_t conn_hdl, uint8_t dir)<br>Remove the input and/or output data path(s) associated with a CIS, CIS configuration, or BIS identified by the Connection_Handle parameter. More... |

## Detailed Description

## Data Structures

| | |
|---|---|
| struct | st_ble_iso_big_param_t<br>big param More... |
| struct | st_ble_iso_big_sync_param_t<br>big sync param More... |
| struct | st_ble_cis_qos<br>CIS channel QoS. More... |
| struct | st_ble_iso_cig_param_t<br>CIG group param. More... |
| struct | st_ble_iso_cis_conn_t<br>CIS stream param. More... |
| struct | st_ble_iso_chan_path<br>ISO channel path param. More... |
| struct | st_ble_iso_sdu_t<br>SDU data structure in SDU input/output flow. More... |
| struct | st_ble_iso_bis_qos_t<br>BIS channel QoS data. More... |
| struct | st_ble_iso_create_big_test_param_t<br>This is the parameters used in R_BLE_ISO_CreateBigTest(). More... |
| struct | st_ble_cis_qos_test_t<br>CIS channel QoS. This is the member variables of st_ble_iso_set_cig_param_test_param_t. More... |
| struct | st_ble_iso_set_cig_param_test_param_t<br>Parameters used in R_BLE_ISO_SetCigParamTest(). More... |

| | |
|---|---|
| struct | st_ble_iso_big_comp_evt_t<br>BIG info of a created BIG. More... |
| struct | st_ble_iso_biginfo_rept_evt_t<br>BIG info report in a periodic adv. More... |
| struct | st_ble_iso_cig_set_evt_t<br>CIS. More... |
| struct | st_ble_iso_cis_req_evt_t<br>CIS reqest from remote device. More... |
| struct | st_ble_iso_cis_qos_t<br>CIS channel QoS data. More... |
| struct | st_ble_iso_cis_est_evt_t<br>Information of CIS that was estabilished. More... |
| struct | st_ble_iso_tx_comp_evt_t<br>Information of ISO SDU that was sent. More... |
| struct | st_ble_iso_test_cnt_info_t<br>ISO test count. More... |
| struct | st_ble_iso_test_end_rept_t<br>ISO test report. More... |
| struct | st_ble_iso_link_quality_info_t<br>ISO link qulity information. More... |
| struct | st_ble_iso_tx_sync_info_t<br>iso TX sync information More... |

| | |
|---|---|
| struct | st_ble_iso_group_hdl_evt_t<br>ISO gourp handle. More... |

## Macros

| | |
|---|---|
| #define | BLE_ISO_PACKING_SEQUENTIAL<br>sequential method of arranging subevents of multiple ISO stream |
| #define | BLE_ISO_PACKING_INTERLEAVED<br>interleaved method of arranging subevents of multiple ISO stream |
| #define | BLE_ISO_FRAMING_UNFRAMED<br>Unframed format for sending ISO PDUs. |
| #define | BLE_ISO_FRAMING_FRAMED<br>Framed format for sending ISO PDUs. |
| #define | BLE_ISO_BROADCAST_CODE_SIZE<br>Broadcast code size in BIG. |
| #define | BLE_ISO_TIMESTAMP_NONE<br>timestamp value when ts_valid is 0 |
| #define | BLE_ISO_SYNC_MSE_AUTO<br>Let controller choose the max subevent. |
| #define | BLE_ISO_CIS_ACCEPT<br>Accept a CIS request. |
| #define | BLE_ISO_CIS_REJECT<br>Reject a CIS request. |
| #define | BLE_ISO_DATA_PATH_HCI<br>Value to set the ISO data path over HCI. |

| | |
|---|---|
| #define | BLE_ISO_DATA_PATH_DIR_INPUT<br>audio datapath directions: App to BLE |
| #define | BLE_ISO_DATA_PATH_DIR_OUTPUT<br>audio datapath directions: BLE to App |
| #define | BLE_ISO_DATA_MAX_PDU<br>maximum number of data octets of ISO Data PDU |
| #define | BLE_ISO_DATA_MAX_SDU<br>maximum number of data octets of ISO Data SDU |

### Enumerations

| | |
|---|---|
| enum | e_ble_iso_evt_t<br>ISO Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_iso_big_param_t

| struct st_ble_iso_big_param_t | | |
|---|---|---|
| big param | | |
| Data Fields | | |
| uint8_t | num_bis | Number channels.<br><br>Maximum number of channels in a single group is BLE_ISO_MAX_GROUP_ISO_COUNT |
| uint32_t | sdu_intv | Channel interval in us.<br><br>Value range BT_ISO_SDU_INTERVAL_MIN - BT_ISO_SDU_INTERVAL_MAX. |
| uint16_t | max_sdu | Maximum size of an SDU.<br><br>Value range 0x0001 to 0x0FFF |
| uint16_t | max_latency | Channel Latency in ms.<br><br>Value range 0x0005 to 0x0FA0 |
| uint8_t | rtn | The number of times that every |

| | | |
|---|---|---|
| | | BIS Data PDU should be retransmitted.<br><br>Value range 0x00 to 0x1E |
| uint8_t | phy | Advertising PHY.<br>value: 0x01 — description: Advertiser PHY is 1M PHY.<br>value: 0x02 — description: Advertiser PHY is 2M PHY.<br>value: 0x03 — description: Advertiser PHY is Coded PHY. |
| uint8_t | packing | Channel packing mode.<br><br>the preferred method of arranging subevents of multiple BISes. BLE_ISO_PACKING_SEQUENTIAL for Sequential or BLE_ISO_PACKING_INTERLEAVED for Interleaved |
| uint8_t | framing | Channel framing mode.<br><br>the format of the BIS Data PDUs. BLE_ISO_FRAMING_UNFRAMED for unframed or BLE_ISO_FRAMING_FRAMED for framed. |
| uint8_t | encryption | Whether or not to encrypt the streams. |
| uint8_t | bcode[ BLE_ISO_BROADCAST_CODE_SIZE] | Broadcast code.<br><br>The code used to derive the session key that is used to encrypt and decrypt BIS payloads. |

◆ **st_ble_iso_big_sync_param_t**

| struct st_ble_iso_big_sync_param_t | | |
|---|---|---|
| big sync param | | |
| Data Fields | | |
| uint8_t | num_bis | Number channels in bis_bitfield.<br><br>Maximum number of channels in a single group is BLE_ISO_MAX_GROUP_ISO_COUNT |
| uint32_t | bis_bitfield | Bitfield of the BISes to sync to<br><br>The BIS indexes start from 0x01, so the lowest allowed bit is bit0 that represents index 0x01. To synchronize to e.g. BIS indexes bit1 and bit2, the bitfield value should be bit0 \| bit1. |
| uint8_t | max_subevents | maximum number of subevents that a Controller should use to receive data payloads in each interval for a BIS. BLE_ISO_SYNC_MSE_AUTO to let controller choose. |
| uint16_t | sync_timeout | |
| uint8_t | encryption | Whether or not to encrypt the streams. |
| uint8_t | bcode[ BLE_ISO_BROADCAST_CODE_SIZE] | Broadcast code.<br><br>The code used to derive the session key that is used to encrypt and decrypt BIS payloads. |

### ◆ st_ble_cis_qos

| struct st_ble_cis_qos | | |
|---|---|---|
| CIS channel QoS. | | |
| Data Fields | | |
| uint16_t | max_sdu_c2p | max sdu size of input and output |
| uint16_t | max_sdu_p2c | |
| uint8_t | rtn_c2p | number of times that a CIS Data PDU should be retransmitted. |
| uint8_t | rtn_p2c | |

| | | |
|---|---|---|
| uint8_t | phy_c2p | Transmitter PHY preference.<br><br>The phy_c2p and phy_p2c field is set to a bitwise OR of the following values. All other values are ignored.<br><br>macro / description<br>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01) — Use 1M PHY for Transmitter PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02) — Use 2M PHY for Transmitter PHY.<br>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04) — Use Coded PHY for Transmitter PHY. |
| uint8_t | phy_p2c | |

◆ **st_ble_iso_cig_param_t**

| struct st_ble_iso_cig_param_t | | |
|---|---|---|
| CIG group param. | | |
| Data Fields | | |
| uint8_t | num_cis | Number channels of CIG.<br><br>Maximum number of channels in a single group is BLE_ISO_MAX_GROUP_ISO_COUNT |
| uint32_t | sdu_intv_c2p | Center to Peripheral Channel interval in us. |
| uint32_t | sdu_intv_p2c | Peripheral to Center Channel interval in us. |
| uint16_t | max_latency_c2p | Center to Peripheral Channel Latency in ms.<br><br>Value range 0x0005 - 0x0FA0 |
| uint16_t | max_latency_p2c | Peripheral to CenterCenter to Peripheral Channel Latency in ms.<br><br>Value range 0x0005 - 0x0FA0 |
| uint8_t | packing | Channel packing mode. |

| | | |
|---|---|---|
| | | The preferred method of arranging subevents of multiple CISes. BLE_ISO_PACKING_SEQUENTIAL for Sequential or BLE_ISO_PACKING_INTERLEAVED for Interleaved |
| uint8_t | framing | Channel framing mode.<br><br>The format of the CIS Data PDUs. BLE_ISO_FRAMING_UNFRAMED for unframed or BLE_ISO_FRAMING_FRAMED for framed. |
| st_ble_cis_qos | cis_qos[BLE_ISO_MAX_GROUP_ISO_COUNT] | qos param. |

### ◆ st_ble_iso_cis_conn_t

| struct st_ble_iso_cis_conn_t | | |
|---|---|---|
| CIS stream param. | | |
| Data Fields | | |
| uint8_t | cig_id | CIG id of this group. |
| uint8_t | num_cis | Number channels of CIG.<br><br>Maximum number of channels in a single group is BLE_ISO_MAX_GROUP_ISO_COUNT |
| uint16_t | cis_hdl[BLE_ISO_MAX_GROUP_ISO_COUNT] | handles for every CIS.<br><br>CIS handles are got from event BLE_ISO_EVENT_CIG_PARAM_SET_COMP. |
| uint16_t | acl_hdl[BLE_ISO_MAX_GROUP_ISO_COUNT] | handles of ACL for every CIS. |

### ◆ st_ble_iso_chan_path

| struct st_ble_iso_chan_path | | |
|---|---|---|
| ISO channel path param. | | |
| Data Fields | | |
| uint8_t | path_dir | path direction.<br><br>Select value from BLE_ISO_DATAPATH_DIR_INPUT or BLE_ISO_DATAPATH_DIR_OUTPUT |

| | | |
|---|---|---|
| uint8_t | path_id | path ID<br><br>Only BLE_ISO_DATA_PATH_HCI is accepted. |
| uint8_t | coding_format | Coding Format.<br><br>Refer Bluetooth SIG Assigned Number( https://www.bluetooth.com/specifications/assigned-numbers). |
| uint16_t | company_id | Company ID. |
| uint16_t | vcodec_id | Vendor-defined Codec ID.<br><br>Shall be ignored if coding_format is not 0xFF |
| uint32_t | delay | Controller Delay. |
| uint8_t | codec_conf_len | Codec Configuration length. Reserved for future use. |
| uint8_t * | p_codec_conf | Pointer to an array containing the Codec Configuration. Reserved for future use. |

### ◆ st_ble_iso_sdu_t

| struct st_ble_iso_sdu_t | | |
|---|---|---|
| SDU data structure in SDU input/output flow.<br><br>Event Code : BLE_ISO_EVENT_ISO_RX_DATA_IND. Also the param of R_BLE_ISO_SendData and R_BLE_ISO_SendDataNoCopy.<br><br>ISO SDU structure | | |
| Data Fields | | |
| uint16_t | conn_hdl | Identifier of the logical channel. Should use BIS or CIS handle for this field. |
| uint16_t | ts_valid | if 0 then timestamp value is invalid. |
| uint32_t | timestamp | timestamp of SDU. Valid when ts_valid is not 0. |
| uint16_t | seq_num | sequence number |
| uint16_t | sdu_len | Length of SDU. |
| uint8_t * | p_sdu_data | SDU data. |

### ◆ st_ble_iso_bis_qos_t

| struct st_ble_iso_bis_qos_t |
|---|
| |

BIS channel QoS data.

| Data Fields | | |
|---|---|---|
| uint8_t | nse | Maximum number of subevents in each isochronous event. |
| uint8_t | bn | The number of new payloads in each BIS event. |
| uint8_t | pto | Offset used for pre-transmissions. |
| uint8_t | irc | The number of times a payload is transmitted in a BIS event. |
| uint16_t | max_pdu | Maximum size, in octets, of the payload. |
| uint16_t | iso_interval | The interval, in microseconds, of periodic SDUs. |

### ◆ st_ble_iso_create_big_test_param_t

struct st_ble_iso_create_big_test_param_t

This is the parameters used in R_BLE_ISO_CreateBigTest().

| Data Fields | | |
|---|---|---|
| uint8_t | num_bis | Total number of BISes in the BIG. |
| uint8_t | sdu_interval[3] | The interval, in microseconds, of periodic SDUs. |
| uint16_t | iso_interval | The time between consecutive BIG anchor points. |
| uint8_t | nse | The total number of subevents in each interval of each BIS in the BIG. |
| uint16_t | max_sdu | Maximum size, in octets, of an SDU. |
| uint16_t | max_pdu | Maximum size, in octets, of payload. |
| uint8_t | phy | The transmitter PHY of packets. |
| uint8_t | packing | The preferred method of arranging subevents of multiple BISes. |
| uint8_t | framing | The format for sending BIS Data PDUs. |
| uint8_t | bn | Number of new payloads for each BIS in a BIS event. |
| uint8_t | irc | Number of times the scheduled |

| | | |
|---|---|---|
| | | data packet is transmitted. |
| uint8_t | pto | Offset in number of ISO_Intervals for pre transmissions of data packets. |
| uint8_t | encryption | Encryption mode of the BISes in the BIG. |
| uint8_t | bcode[ BLE_ISO_BROADCAST_CODE_SIZE] | Used to generate the session key to encrypt payloads of all BISes in the BIG. |

### ◆ st_ble_cis_qos_test_t

| struct st_ble_cis_qos_test_t | | |
|---|---|---|
| CIS channel QoS. This is the member variables of st_ble_iso_set_cig_param_test_param_t. | | |
| Data Fields | | |
| uint8_t | cis_id | Used to identify the CIS. |
| uint8_t | nse | Maximum number of subevents for each CIS in a CIG event. |
| uint16_t | c_sdu | Maximum size, in octets, of the payload from the Central Host. |
| uint16_t | p_sdu | Maximum size, in octets, of the payload from the Peripheral Host. |
| uint16_t | c_pdu | Maximum size, in octets, of the payload from the Central Link Layer to the Peripheral Link Layer. |
| uint16_t | p_pdu | Maximum size, in octets, of the payload from the Peripheral Link Layer to the Central Link Layer. |
| uint8_t | c_phy | The transmitter PHY of packets from the Central. |
| uint8_t | p_phy | The transmitter PHY of packets from the Peripheral. |
| uint8_t | c_bn | Burst number for Central to Peripheral. |
| uint8_t | p_bn | Burst number for Peripheral to Central. |

### ◆ st_ble_iso_set_cig_param_test_param_t

| struct st_ble_iso_set_cig_param_test_param_t |
|---|
| Parameters used in R_BLE_ISO_SetCigParamTest(). |
| Data Fields |

| | | |
|---|---|---|
| uint8_t | c_interval[3] | Time interval of periodic SDUs from the Central Host. |
| uint8_t | p_interval[3] | Time interval of periodic SDUs from the Peripheral Host. |
| uint8_t | c_ft | Maximum time for a payload from the Central to Peripheral to be transmitted and re-transmitted, after which it is flushed. |
| uint8_t | p_ft | Maximum time for a payload from the Peripheral to Central to be transmitted and re-transmitted, after which it is flushed. |
| uint16_t | iso_interval | Time between two consecutive CIS anchor points. |
| uint8_t | sca | Worst-case sleep clock accuracy of all the Peripherals that will participate in the CIG. |
| uint8_t | packing | Preferred method of arranging subevents of multiple CISes. |
| uint8_t | framing | Format of the CIS Data PDUs of all the CISes. |
| uint8_t | num_cis | Total number of CIS configurations in the CIG being added or modified. |
| st_ble_cis_qos_test_t | cis[BLE_ISO_MAX_GROUP_ISO_COUNT] | CIS QoS configurations. |

### ◆ st_ble_iso_big_comp_evt_t

| struct st_ble_iso_big_comp_evt_t | | |
|---|---|---|
| BIG info of a created BIG.<br>Event Code : BLE_ISO_EVENT_CREATE_BIG_COMP BLE_ISO_EVENT_CREATE_BIG_SYNC_COMP: st_ble_iso_big_comp_evt_t | | |
| Data Fields | | |
| uint8_t | status | status of BIG creation |
| uint8_t | big_hdl | BIG handle. |
| uint32_t | sync_delay | BIG sync delay. |
| uint32_t | latency | Actual latency returned by controller. |
| uint8_t | phy | PHY. |
| st_ble_iso_bis_qos_t | bis_qos | qos parameters |

| | | |
|---|---|---|
| uint8_t | num_bis | number of streams in the group |
| uint16_t | bis_hdl[BLE_ISO_MAX_GROUP_ISO_COUNT] | handles of all streams |

### ◆ st_ble_iso_biginfo_rept_evt_t

| struct st_ble_iso_biginfo_rept_evt_t | | |
|---|---|---|
| BIG info report in a periodic adv. | | |
| Data Fields | | |
| uint8_t | sid | Advertiser SID. |
| uint8_t | num_bis | Number of BISes in the BIG. |
| uint8_t | sub_evt_count | Maximum number of subevents in each isochronous event. |
| uint16_t | iso_interval | Interval between two BIG anchor point (N * 1.25 ms) |
| uint8_t | burst_number | The number of new payloads in each BIS event. |
| uint8_t | pto | Offset used for pre-transmissions. |
| uint8_t | rep_count | The number of times a payload is transmitted in a BIS event. |
| uint16_t | max_pdu | Maximum size, in octets, of the payload. |
| uint32_t | sdu_interval | The interval, in microseconds, of periodic SDUs. |
| uint16_t | max_sdu | Maximum size of an SDU, in octets. |
| uint8_t | phy | Channel PHY. |
| uint8_t | framing | Channel framing mode. |
| uint8_t | encryption | Whether or not the BIG is encrypted. |

### ◆ st_ble_iso_cig_set_evt_t

| struct st_ble_iso_cig_set_evt_t | | |
|---|---|---|
| CIS. | | |
| Data Fields | | |
| uint8_t | cig_id | number of CIG streams |
| uint8_t | num_cis | number of CIS streams |
| uint8_t | cis_id[BLE_ISO_MAX_GROUP_ISO_COUNT] | cis_id over peers. Application should store cis_id for profile's usage |

| | | |
|---|---|---|
| uint16_t | cis_hdl[BLE_ISO_MAX_GROUP_ISO_COUNT] | CIS conn handle. |

### ◆ st_ble_iso_cis_req_evt_t

| struct st_ble_iso_cis_req_evt_t | | |
|---|---|---|
| CIS reqest from remote device. | | |
| Data Fields | | |
| uint8_t | cig_id | cig_id over peers. Application should check the request by cig_id and cis_id |
| uint8_t | cis_id | cis_id over peers. Application should check the request by cig_id and cis_id |
| uint16_t | acl_hdl | The ACL handle over which this cis is created. |

### ◆ st_ble_iso_cis_qos_t

| struct st_ble_iso_cis_qos_t | | |
|---|---|---|
| CIS channel QoS data. | | |
| Data Fields | | |
| uint8_t | nse | Maximum number of subevents in each isochronous event. |
| uint8_t | bn_c2p | The number of new payloads in each CIS event. |
| uint8_t | bn_p2c | |
| uint8_t | flush_to_c2p | The flush timeout, in multiples of the ISO_Interval for the CIS. |
| uint8_t | flush_to_p2c | |
| uint16_t | max_pdu_c2p | Maximum size, in octets, of the payload. |
| uint16_t | max_pdu_p2c | |
| uint16_t | iso_interval | The time between two consecutive CIS anchor points. |

### ◆ st_ble_iso_cis_est_evt_t

| struct st_ble_iso_cis_est_evt_t | | |
|---|---|---|
| Information of CIS that was estabilished. | | |
| Data Fields | | |
| uint16_t | cis_hdl | CIS handler. |
| uint32_t | cig_sync_delay | CIG sync delay. |

| | | |
|---|---|---|
| uint32_t | cis_sync_delay | CIS sync delay. |
| uint32_t | latency_c2p | latency of each direction |
| uint32_t | latency_p2c | |
| uint8_t | phy_c2p | PHY of each direction. |
| uint8_t | phy_p2c | |
| st_ble_iso_cis_qos_t | cis_qos | qos parameters |

### ◆ st_ble_iso_tx_comp_evt_t

| struct st_ble_iso_tx_comp_evt_t | | |
|---|---|---|
| Information of ISO SDU that was sent. | | |
| Data Fields | | |
| uint16_t | conn_hdl | connection handle of the ISO packet that was sent |
| uint32_t | seq_num | sequence number of ISO packet that was sent |

### ◆ st_ble_iso_test_cnt_info_t

| struct st_ble_iso_test_cnt_info_t | | |
|---|---|---|
| ISO test count. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Indicate an BIS or CIS. |
| uint32_t | received_cnt | Number in the Received_SDU_Count. |
| uint32_t | missed_cnt | Number in the Missed_SDU_Count. |
| uint32_t | failed_cnt | Number in the Failed_SDU_Count. |

### ◆ st_ble_iso_test_end_rept_t

| struct st_ble_iso_test_end_rept_t | | |
|---|---|---|
| ISO test report. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Indicate an BIS or CIS. |
| uint32_t | received_cnt | Number in the Received_SDU_Count. |
| uint32_t | missed_cnt | Number in the Missed_SDU_Count. |
| uint32_t | failed_cnt | Number in the Failed_SDU_Count. |

### ◆ st_ble_iso_link_quality_info_t

| struct st_ble_iso_link_quality_info_t | | |
|---|---|---|
| ISO link qulity information. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Indicate an BIS or CIS. |
| uint32_t | tx_unacked_packets | Value of the Tx_UnACKed_Packets counter. |
| uint32_t | tx_flushed_packets | Value of the Tx_Flushed_Packets counter. |
| uint32_t | tx_last_subevent_packets | Value of the Tx_Last_Subevent_Packets counter. |
| uint32_t | retransmitted_packets | Value of the Retransmitted_Packets counter. |
| uint32_t | crc_error_packets | Value of the CRC_Error_Packets counter. |
| uint32_t | rx_unreceived_packets | Value of the Rx_Unreceived_Packets counter. |
| uint32_t | duplicate_packets | Value of the Duplicate_Packets counter. |

### ◆ st_ble_iso_tx_sync_info_t

| struct st_ble_iso_tx_sync_info_t | | |
|---|---|---|
| iso TX sync information | | |
| Data Fields | | |
| uint16_t | conn_hdl | Indicate an BIS or CIS. |
| uint32_t | ts | CIG reference point or BIG anchor point of a transmitted SDU, in microseconds. |
| uint32_t | offset | Time offset, in microseconds |
| uint16_t | seq_num | Packet sequence number |

### ◆ st_ble_iso_group_hdl_evt_t

| struct st_ble_iso_group_hdl_evt_t | | |
|---|---|---|
| ISO gourp handle. | | |
| Data Fields | | |
| uint8_t | group_hdl | BIG handle or CIG id. |

## Enumeration Type Documentation

### ◆ e_ble_iso_evt_t

| enum e_ble_iso_evt_t | |
|---|---|
| ISO Event Identifier. | |
| Enumerator | |
| BLE_ISO_EVENT_CREATE_BIG_COMP | R_BLE_ISO_CreateBig() created a big successfully.<br>**Event Data:**<br>st_ble_iso_big_comp_evt_t |
| BLE_ISO_EVENT_BIGINFO_REPT | Big Info is detected in a periodic adv data.<br>**Event Data:**<br>st_ble_iso_biginfo_rept_evt_t |
| BLE_ISO_EVENT_CREATE_BIG_SYNC_COMP | The sync-id specified by R_BLE_ISO_CreateBigSync() has already established a big sync.<br>st_ble_iso_big_comp_evt_t |
| BLE_ISO_EVENT_ISO_RX_DATA_IND | An ISO SDU is received.<br>**Event Data:**<br>st_ble_iso_sdu_t |
| BLE_ISO_EVENT_ISO_TX_COMP | An ISO SDU is sent.<br>**Event Data:**<br>st_ble_iso_tx_comp_evt_t |
| BLE_ISO_EVENT_SYNC_LOST | This event notifies the application layer that the ISO sync has failed.<br>**Event Data:**<br>st_ble_iso_group_hdl_evt_t |
| BLE_ISO_EVENT_SYNC_TERM | This event notifies the application layer that the BIG sync has been terminated.<br>**Event Data:**<br>st_ble_iso_group_hdl_evt_t |
| BLE_ISO_EVENT_CIG_PARAM_SET_COMP | The request for CIG setting parameter has been sent to Controller. |

| | |
|---|---|
| | **Event Data:**<br>st_ble_iso_cig_set_evt_t |
| BLE_ISO_EVENT_CIS_REQ | The CIS request has been received from to remote device.<br>**Event Data:**<br>st_ble_iso_cis_req_evt_t |
| BLE_ISO_EVENT_CIS_EST | An CIS connection is estabilished.<br>**Event Data:**<br>st_ble_iso_cis_est_evt_t |
| BLE_ISO_EVENT_SETUP_DATA_PATH_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_ISO_EVENT_CIG_REMOVE_COMP | **Event Data:**<br>st_ble_iso_group_hdl_evt_t |
| BLE_ISO_EVENT_BIG_REMOVE_COMP | **Event Data:**<br>st_ble_iso_group_hdl_evt_t |
| BLE_ISO_EVENT_REPLY_CIS_REQ_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_ISO_EVENT_GET_TX_SYNC_COMP | **Event Data:**<br>st_ble_iso_tx_sync_info_t |
| BLE_GAP_EVENT_READ_ANT_INFO_COMP | **Event Data:**<br>st_ble_gap_cte_antenna_info_t |
| BLE_ISO_EVENT_TX_TEST_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_ISO_EVENT_RX_TEST_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_ISO_EVENT_READ_TEST_CNT_COMP | **Event Data:**<br>st_ble_iso_test_cnt_info_t |
| BLE_ISO_EVENT_TEST_ENDED | **Event Data:** |

| | |
|---|---|
| | st_ble_iso_test_end_rept_t |
| BLE_ISO_EVENT_READ_LINK_QUALITY_COMP | **Event Data:**<br>st_ble_iso_link_quality_info_t |
| BLE_ISO_EVENT_REMOVE_DATAPATH_COMP | **Event Data:**<br>st_ble_gap_conn_hdl_evt_t |
| BLE_GAP_EVENT_PER_ADV_RECV_ON | **Event Data:**<br>None |
| BLE_GAP_EVENT_PER_ADV_RECV_OFF | **Event Data:**<br>None |

## Function Documentation

### ◆ R_BLE_ISO_CreateBig()

ble_status_t R_BLE_ISO_CreateBig ( uint8_t * *p_big_hdl*, uint8_t *adv_hdl*, st_ble_iso_big_param_t * *p_big_param* )

Create a BIG.

**Parameters**

| | | |
|---|---|---|
| [out] | p_big_hdl | BIG handle is assigned by host stack, and value is passed out. |
| [in] | adv_hdl | Periodic adv handle which carries the biginfo. |
| [in] | p_big_param | BIG param |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_StopBig()

ble_status_t R_BLE_ISO_StopBig ( uint8_t *big_hdl*, uint8_t *reason* )

Stop a BIG of the big_handle.

**Parameters**

| | | |
|---|---|---|
| [in] | big_hdl | handle of the BIG to be stopped. |
| [in] | reason | reason of termination. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_CreateBigSync()

ble_status_t R_BLE_ISO_CreateBigSync ( uint8_t * *p_big_hdl*, uint16_t *sync_hdl*, st_ble_iso_big_sync_param_t * *p_big_sync_param* )

Create a BIG sync.

**Parameters**

| | | |
|---|---|---|
| [out] | p_big_hdl | BIG handle is assigned by host stack, and value is passed out. |
| [in] | sync_hdl | Periodic adv sid which carries the biginfo. |
| [in] | p_big_sync_param | BIG param which is got from biginfo. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_TerminateBigSync()

| ble_status_t R_BLE_ISO_TerminateBigSync ( uint8_t *big_hdl*) |
|---|

Terminate a BIG sync.

**Parameters**

| | | |
|---|---|---|
| [in] | big_hdl | handle of the BIG sync to be terminated. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_SetCigParam()

| ble_status_t R_BLE_ISO_SetCigParam ( uint8_t * *p_cig_id*, st_ble_iso_cig_param_t * *p_cig_param* ) |
|---|

Create a CIG with param.

This function requests BLE system to create a CIG. The result of this API call is notified in BLE_ISO_EVENT_CIG_PARAM_SET_COMP event.

**Parameters**

| | | |
|---|---|---|
| [out] | p_cig_id | CIG is assigned by host stack, and value is passed out. |
| [in] | p_cig_param | CIG param |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_CreateCis()

ble_status_t R_BLE_ISO_CreateCis ( st_ble_iso_cis_conn_t * *p_cis_conn*)

Create one or more CISes using the CIS param.

This function send the CIS requests to the remote devices. The result of this API call is returned by a return value. Remote device receives BLE_ISO_EVENT_CIS_REQ event. The response from remote device is notified in BLE_ISO_EVENT_CIS_EST event. Once CIS connectionis estabilished successfully, they can be disconnected by R_BLE_GAP_Disconnect.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cis_conn | CIS param |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_RemoveCig()

ble_status_t R_BLE_ISO_RemoveCig ( uint8_t *cig_id*)

remove a CIG of id and all of CIS streams in this CIG.

**Parameters**

| | | |
|---|---|---|
| [out] | cig_id | CIG id to be removed. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_ReplyCisRequest()

ble_status_t R_BLE_ISO_ReplyCisRequest ( uint8_t *cig_id*, uint8_t *cis_id*, uint8_t *response*, uint8_t *reason* )

Reply the CIS request from a remote device.

This function replies to the CIS request from the remote device. The CIS request from the remote device is notified in BLE_ISO_EVENT_CIS_REQ event. The result of this API call is returned by a return value. The result is notified in BLE_ISO_EVENT_CIS_EST event.

**Parameters**

| | | |
|---|---|---|
| [in] | cig_id | CIG ID |
| [in] | cis_id | CIS ID |
| [in] | response | Accept or reject the pairing request from the remote device.<br>macro: BLE_ISO_CIS_ACCEPT(0x00) — Accept the CIS request.<br>BLE_ISO_CIS_REJECT(0x01) — Reject the CIS request. |
| [in] | reason | The reason for rejecting CIS request. This parameter is ignored when response is BLE_ISO_CIS_ACCEPT. Refer the error code described in Core Specification Vol.2 Part D ,"2 Error Code Descriptions". |

### ◆ R_BLE_ISO_SetupDataPath()

ble_status_t R_BLE_ISO_SetupDataPath ( uint16_t *conn_hdl*, st_ble_iso_chan_path * *p_path* )

Create the ISO data path between the Host and the Controller for a CIS.

This function behaviors depending on the platform. For SoC without Bluetooth Audio support, only the following value are valid in param path:

| field | value |
|---|---|
| path_id | BLE_ISO_DATA_PATH_HCI |
| coding_format | 0xFF (Vendor specific coding format) |

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | CIS handle |
| [in] | p_path | data path configuration |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_SendData()

ble_status_t R_BLE_ISO_SendData ( st_ble_iso_sdu_t * *p_sdu_info*)

Send a SDU payload to a ISO channel of conn_hdl.

This function copies SDU payload into host's buffer and push the buffer into host's TX queue. Once the payload is accepted by controller scheduler, BLE_ISO_EVENT_ISO_TX_COMP event is sent back to application. Due to limitation of host buffer capacity, the maximum size of SDU payload is 251 bytes (BLE_ISO_DATA_MAX_PDU).

**Parameters**

| | | |
|---|---|---|
| [in] | p_sdu_info | SDU info and data |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_SendDataNoCopy()

ble_status_t R_BLE_ISO_SendDataNoCopy ( st_ble_iso_sdu_t * *p_sdu_info*)

Send SDU payload to a ISO channel of conn_hdl without copying data.

This function behaviors similar to R_BLE_ISO_SendData, but without copying SDU payload to the buffer in host. For this reason application should hold the buffer until a BLE_ISO_EVENT_ISO_TX_COMP event with the same conn_hdl and seq_number of p_sdu_info is received. The maximum size of SDU payload defined by BLE_ISO_DATA_MAX_SDU.

**Parameters**

| | | |
|---|---|---|
| [in] | p_sdu_info | SDU info and data |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_ISO_CreateBigTest()

ble_status_t R_BLE_ISO_CreateBigTest ( uint8_t * *p_big_hdl*, uint8_t *adv_hdl*, st_ble_iso_create_big_test_param_t * *p_create_big_test_param* )

Create one or more BISes of a BIG (see [Vol 6] Part B, Section 4.4.6). All BISes in the BIG have the same values for all parameters.

**Parameters**

| | | |
|---|---|---|
| [out] | p_big_hdl | Used to identify a BIG. |
| [in] | adv_hdl | Used to identify an advertising set. |
| [in] | p_create_big_test_param | specified parameter of BIG param test |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_SetCigParamTest()

ble_status_t R_BLE_ISO_SetCigParamTest ( uint8_t * *p_cig_id*,
st_ble_iso_set_cig_param_test_param_t * *p_set_cig_param_test_param* )

Create a CIG and set the parameters of one or more CISes that are associated with a CIG in the Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_set_cig_param_test_param | specified parameter of CIG param_test |
| [out] | p_cig_id | cig ID is assigned by host stack, and value is passed out. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_TransmitTest()

ble_status_t R_BLE_ISO_TransmitTest ( uint16_t *conn_hdl*, uint8_t *payload_type* )

Configure an established CIS or BIS and transmit test payloads which are generated by the Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS. |
| [in] | payload_type | Configuration of SDUs in the payload. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_ReceiveTest()

ble_status_t R_BLE_ISO_ReceiveTest ( uint16_t *conn_hdl*, uint8_t *payload_type* )

Configure an established CIS or a synchronized BIG to receive payloads.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS. |
| [in] | payload_type | Configuration of SDUs in the payload. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_ReadTestCounters()

ble_status_t R_BLE_ISO_ReadTestCounters ( uint16_t *conn_hdl*)

Read the test counters (see [Vol 6] Part B, Section 7) in the Controller which is configured in ISO Receive Test mode for a CIS or BIS specified by the Connection_Handle. Reading the test counters does not reset the test counters.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_TestEnd()

ble_status_t R_BLE_ISO_TestEnd ( uint16_t *conn_hdl*)

Terminate the ISO Transmit and/or Receive Test mode for a CIS or BIS specified by the Connection_Handle parameter but does not terminate the CIS or BIS.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_ReadLinkQuality()

ble_status_t R_BLE_ISO_ReadLinkQuality ( uint16_t *conn_hdl*)

Returns the values of various counters related to link quality that are associated with the isochronous stream specified by the Connection_Handle parameter.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success, otherwise "HCI Spec Error" in enum RBLE_STATUS_enum. |

### ◆ R_BLE_ISO_RemoveDataPath()

ble_status_t R_BLE_ISO_RemoveDataPath ( uint16_t *conn_hdl*, uint8_t *dir* )

Remove the input and/or output data path(s) associated with a CIS, CIS configuration, or BIS identified by the Connection_Handle parameter.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle of the CIS or BIS |
| [in] | dir | Data path direction. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_CHAN(0x0005) | The handle does not indicate a CIS or BIS channel. |
| BLE_ERR_UNSPECIFIED(0x0013) | Unspecified error. |

## GATT_COMMON

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_GATT_GetMtu (uint16_t conn_hdl, uint16_t *p_mtu)<br>This function gets the current MTU used in GATT communication. More... |

### Detailed Description

### Function Documentation

### ◆ R_BLE_GATT_GetMtu()

ble_status_t R_BLE_GATT_GetMtu ( uint16_t *conn_hdl*, uint16_t * *p_mtu* )

This function gets the current MTU used in GATT communication.

Both GATT server and GATT Client can use this function.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server or the GATT Client. |
| [in] | p_mtu | The Current MTU. Before MTU exchange, this parameter is 23 bytes. After MTU exchange, this parameter is the negotiated MTU. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The mtu parameter is NULL. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server or the GATT Client specified by conn_hdl was not found. |

## GATT_SERVER

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_GATTS_Init (uint8_t cb_num)<br>This function initializes the GATT Server and registers the number of the callbacks for GATT Server event. More... |
| ble_status_t | R_BLE_GATTS_SetDbInst (st_ble_gatts_db_cfg_t *p_db_inst)<br>This function sets GATT Database to host stack. More... |
| ble_status_t | R_BLE_GATTS_RegisterCb (ble_gatts_app_cb_t cb, uint8_t priority)<br>This function registers a callback for GATT Server event. More... |
| ble_status_t | R_BLE_GATTS_DeregisterCb (ble_gatts_app_cb_t cb) |

| | |
|---|---|
| | This function deregisters the callback function for GATT Server event. More... |
| ble_status_t | R_BLE_GATTS_Notification (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_ntf_data)<br>This function sends a notification of an attribute's value. More... |
| ble_status_t | R_BLE_GATTS_Indication (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_ind_data)<br>This function sends a indication of an attribute's value. More... |
| ble_status_t | R_BLE_GATTS_GetAttr (uint16_t conn_hdl, uint16_t attr_hdl, st_ble_gatt_value_t *p_value)<br>This function gets a attribute value from the GATT Database. More... |
| ble_status_t | R_BLE_GATTS_SetAttr (uint16_t conn_hdl, uint16_t attr_hdl, st_ble_gatt_value_t *p_value)<br>This function sets an attribute value to the GATT Database. More... |
| ble_status_t | R_BLE_GATTS_SendErrRsp (uint16_t error_code)<br>This function sends an error response to a remote device. More... |
| ble_status_t | R_BLE_GATTS_RspExMtu (uint16_t conn_hdl, uint16_t mtu)<br>This function replies to a MTU Exchange Request from a remote device. More... |
| ble_status_t | R_BLE_GATTS_SetPrepareQueue (st_ble_gatt_pre_queue_t *p_pre_queues, uint8_t queue_num)<br>Register prepare queue and buffer in Host Stack. More... |

## Detailed Description

### Data Structures

| | |
|---|---|
| struct | st_ble_gatt_value_t<br>Attribute Value. More... |

| | |
|---|---|
| struct | st_ble_gatt_hdl_value_pair_t<br>Attribute handle and attribute Value. More... |
| struct | st_ble_gatt_queue_att_val_t<br>Queued writes Attribute Value. More... |
| struct | st_ble_gatt_queue_pair_t<br>Queued writes Attribute Value. More... |
| struct | st_ble_gatt_queue_elm_t<br>Prepare Write Queue element for long chracteristic. More... |
| struct | st_ble_gatt_pre_queue_t<br>Prepare Write Queue for long chracteristic. More... |
| struct | st_ble_gatts_db_params_t<br>Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client. More... |
| struct | st_ble_gatts_db_conn_hdl_t<br>Information about the service or the characteristic that the attribute belongs to. More... |
| struct | st_ble_gatts_db_access_evt_t<br>This structure notifies that the GATT Database has been accessed from a GATT Client. More... |
| struct | st_ble_gatts_conn_evt_t<br>This structure notifies that the link with the GATT Client has been established. More... |
| struct | st_ble_gatts_disconn_evt_t<br>This structure notifies that the link with the GATT Client has been disconnected. More... |
| struct | st_ble_gatts_ex_mtu_req_evt_t |

| | |
|---|---|
| | This structure notifies that a MTU Exchange Request PDU has been received from a GATT Client. More... |
| struct | st_ble_gatts_cfm_evt_t<br>This structure notifies that a Confirmation PDU has been received from a GATT Client. More... |
| struct | st_ble_gatts_read_by_type_rsp_evt_t<br>This structure notifies that a Read By Type Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_read_rsp_evt_t<br>This structure notifies that a Read Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_read_blob_rsp_evt_t<br>This structure notifies that a Read Blob Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_read_multi_rsp_evt_t<br>This structure notifies that a Read Multiple Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_write_rsp_evt_t<br>This structure notifies that a Write Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_prepare_write_rsp_evt_t<br>This structure notifies that a Prepare Write Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_exe_write_rsp_evt_t<br>This structure notifies that a Execute Write Response PDU has been sent from GATT Server. More... |
| struct | st_ble_gatts_db_uuid_cfg_t<br>A structure that defines the information on the position where UUIDs |

| | |
|---|---|
| | are used. More... |
| struct | st_ble_gatts_db_attr_cfg_t<br>A structure that defines the detailed information of the attributes. More... |
| struct | st_ble_gatts_db_attr_list_t<br>The number of attributes are stored. More... |
| struct | st_ble_gatts_db_char_cfg_t<br>A structure that defines the detailed information of the characteristics. More... |
| struct | st_ble_gatts_db_serv_cfg_t<br>A structure that defines the detailed information of the characteristics. More... |
| struct | st_ble_gatts_db_cfg_t<br>This is the structure of GATT Database that is specified in R_BLE_GATTS_SetDbInst(). More... |
| struct | st_ble_gatts_evt_data_t<br>st_ble_gatts_evt_data_t is the type of the data notified in a GATT Server Event. More... |

## Macros

| | |
|---|---|
| #define | BLE_GATT_DEFAULT_MTU<br>GATT Default MTU. |
| #define | BLE_GATT_16_BIT_UUID_FORMAT<br>GATT Identification for 16-bit UUID Format. |
| #define | BLE_GATT_128_BIT_UUID_FORMAT<br>GATT Identification for 128-bit UUID Format. |
| #define | BLE_GATT_16_BIT_UUID_SIZE |

| | |
|---|---|
| | GATT 16-bit UUID Size. |
| #define | BLE_GATT_128_BIT_UUID_SIZE<br>GATT 128-bit UUID Size. |
| #define | BLE_GATT_INVALID_ATTR_HDL_VAL<br>GATT Invalid Attribute Handle Value. |
| #define | BLE_GATT_ATTR_HDL_START_RANGE<br>GATT Attribute Handle Start Range. |
| #define | BLE_GATT_ATTR_HDL_END_RANGE<br>GATT Attribute Handle End Range. |
| #define | BLE_GATTS_CLI_CNFG_NOTIFICATION<br>GATT Client Configuration values. Enable Notification. |
| #define | BLE_GATTS_CLI_CNFG_INDICATION<br>GATT Client Configuration values. Enable Indication. |
| #define | BLE_GATTS_CLI_CNFG_DEFAULT<br>GATT Client Configuration values. Default value or disable notification/indication. |
| #define | BLE_GATTS_SER_CNFG_BROADCAST<br>GATT Server Configuration values. Enable broadcast. |
| #define | BLE_GATTS_SER_CNFG_DEFAULT<br>GATT Server Configuration values. Default value. |
| #define | BLE_GATTS_MAX_CB<br>GATT Server Callback Number. |
| #define | BLE_GATTS_OP_CHAR_VALUE_READ_REQ |

| | |
|---|---|
| | Characteristic Value Local Read Operation. |
| #define | BLE_GATTS_OP_CHAR_VALUE_WRITE_REQ<br>Characteristic Value Local Write Operation. |
| #define | BLE_GATTS_OP_CHAR_VALUE_WRITE_WITHOUT_REQ<br>Characteristic Value Local Write Without Response Operation. |
| #define | BLE_GATTS_OP_CHAR_CLI_CNFG_READ_REQ<br>Characteristic Client Configuration Local Read Operation. |
| #define | BLE_GATTS_OP_CHAR_CLI_CNFG_WRITE_REQ<br>Characteristic Client Configuration Local Write Operation. |
| #define | BLE_GATTS_OP_CHAR_SER_CNFG_READ_REQ<br>Characteristic Server Configuration Local Read Operation. |
| #define | BLE_GATTS_OP_CHAR_SER_CNFG_WRITE_REQ<br>Characteristic Server Configuration Local Write Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_READ_REQ<br>Characteristic Value Peer Read Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_WRITE_REQ<br>Characteristic Value Peer Write Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_WRITE_CMD<br>Characteristic Value Peer Write Command. |
| #define | BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_READ_REQ<br>Characteristic Client Configuration Peer Read Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_WRITE_REQ<br>Characteristic Client Configuration Peer Write Operation. |

| | |
|---|---|
| #define | BLE_GATTS_OP_CHAR_PEER_SER_CNFG_READ_REQ<br>Characteristic Server Configuration Peer Read Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_SER_CNFG_WRITE_REQ<br>Characteristic Server Configuration Peer Write Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_USR_DESC_READ_REQ<br>Characteristic User Description Peer Read Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_USR_DESC_WRITE_REQ<br>Characteristic User Description Peer Write Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_HLD_DESC_READ_REQ<br>Characteristic Higher Layer Defined Descriptor Peer Read Operation. |
| #define | BLE_GATTS_OP_CHAR_PEER_HLD_DESC_WRITE_REQ<br>Characteristic Higher Layer Defined Descriptor Peer Write Operation. |
| #define | BLE_GATTS_OP_CHAR_REQ_AUTHOR<br>Operation Required Authorization. |
| #define | BLE_GATT_DB_READ<br>Allow clients to read. |
| #define | BLE_GATT_DB_WRITE<br>Allow clients to write. |
| #define | BLE_GATT_DB_WRITE_WITHOUT_RSP<br>Allow clients to write without response. |
| #define | BLE_GATT_DB_READ_WRITE<br>Allow clients to access of all. |

#define BLE_GATT_DB_NO_AUXILIARY_PROPERTY

No auxiliary properties.

#define BLE_GATT_DB_FIXED_LENGTH_PROPERTY

Fixed length attribute value.

#define BLE_GATT_DB_AUTHORIZATION_PROPERTY

Attributes requiring authorization.

#define BLE_GATT_DB_ATTR_DISABLED

The attribute is disabled. If this value is set, the attribute cannot be found and accessed by a GATT Client.

#define BLE_GATT_DB_128_BIT_UUID_FORMAT

Attribute with 128 bit UUID.

#define BLE_GATT_DB_PEER_SPECIFIC_VAL_PROPERTY

Attribute managed by each GATT Client.

#define BLE_GATT_DB_CONST_ATTR_VAL_PROPERTY

Fixed attribute value.

#define BLE_GATT_DB_SER_SECURITY_UNAUTH

Unauthenticated pairing(Security Mode1 Security Level 2, Security Mode 2 Security Level 1). Unauthenticated pairing is required to access the service.

#define BLE_GATT_DB_SER_SECURITY_AUTH

Authenticated pairing(Security Mode1 Security Level 3, Security Mode 2 Security Level 2). Authenticated pairing is required to access the service.

#define BLE_GATT_DB_SER_SECURITY_SECONN

Authenticated LE secure connections that generates 16bytes LTK(Security Mode1 Security Level 4). Authenticated LE secure connections pairing that generates 16bytes LTK is required to access the service. If this bit is set, bit24-27 are ignored.

| | |
|---|---|
| #define | BLE_GATT_DB_SER_SECURITY_ENC<br>Encryption. Encryption by the LTK exchanged in pairing is required to access. |
| #define | BLE_GATT_DB_SER_NO_SECURITY_PROPERTY<br>No Security(Security Mode1 Security Level 1). |
| #define | BLE_GATT_DB_SER_ENC_KEY_SIZE_DONT_CARE<br>7-byte or larger encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_7<br>7-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_8<br>8-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_9<br>9-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_10<br>10-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_11<br>11-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_12<br>12-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_13<br>13-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_14<br>14-byte encryption key. |

| | |
|---|---|
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_15<br>15-byte encryption key. |
| #define | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_16<br>16-byte encryption key. |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_gatts_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data)<br>ble_gatts_app_cb_t is the GATT Server Event callback function type. More... |

### Enumerations

| | |
|---|---|
| enum | e_r_ble_gatts_evt_t<br>GATT Server Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_gatt_value_t

| struct st_ble_gatt_value_t | | |
|---|---|---|
| Attribute Value. | | |
| Data Fields | | |
| uint16_t | value_len | Length of the attribute value. |
| uint8_t * | p_value | Attribute Value. |

### ◆ st_ble_gatt_hdl_value_pair_t

| struct st_ble_gatt_hdl_value_pair_t | | |
|---|---|---|
| Attribute handle and attribute Value. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute Handle. |
| st_ble_gatt_value_t | value | Attribute Value. |

### ◆ st_ble_gatt_queue_att_val_t

| struct st_ble_gatt_queue_att_val_t |
|---|
| Queued writes Attribute Value. |
| Data Fields |

| | | |
|---|---|---|
| uint8_t * | p_value | Attribute Value for Queued Write . |
| uint16_t | value_len | Length of the attribute value. |
| uint16_t | padding | padding. |

### ◆ st_ble_gatt_queue_pair_t

| struct st_ble_gatt_queue_pair_t | | |
|---|---|---|
| Queued writes Attribute Value. | | |
| Data Fields | | |
| st_ble_gatt_queue_att_val_t | queue_value | Attribute Value for Queued Write. |
| uint16_t | attr_hdl | Attribute Handle. |

### ◆ st_ble_gatt_queue_elm_t

| struct st_ble_gatt_queue_elm_t | | |
|---|---|---|
| Prepare Write Queue element for long chracteristic. | | |
| Data Fields | | |
| st_ble_gatt_queue_pair_t | queue_value_pair | Part of Long Characteristic Value and Characteristic Value Handle. |
| uint16_t | offset | Offset that indicates the location to be written. |

### ◆ st_ble_gatt_pre_queue_t

| struct st_ble_gatt_pre_queue_t | | |
|---|---|---|
| Prepare Write Queue for long chracteristic. | | |
| Data Fields | | |
| uint8_t * | p_buf_start | Buffer start address for Write Long Characteristic Request. |
| st_ble_gatt_queue_elm_t * | p_queue | Prepare Write Queue for Long Characteristic Value. |
| uint16_t | buffer_len | Buffer length. |
| uint16_t | conn_hdl | Connection Handle. |
| uint16_t | buf_offset | Current buffer offset. |
| uint8_t | queue_size | Number of elements in the prepare write queue. |
| uint8_t | queue_idx | Index of Prepare Write Queue. |

### ◆ st_ble_gatts_db_params_t

| struct st_ble_gatts_db_params_t |
|---|

| Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client. | | |
|---|---|---|
| Data Fields | | |
| st_ble_gatt_value_t | value | Attribute value to be set to or retrieved from the GATT Database. Note that the address of the value field in the value field is invalid in case of read access. |
| uint16_t | attr_hdl | Attribute handle identifying the attribute to be set or retrieved. |
| uint8_t | db_op | Type of the access to GATT Database from the GATT Client.<br><br>See also<br>access_type_to_gatt_database |

### ◆ st_ble_gatts_db_conn_hdl_t

| struct st_ble_gatts_db_conn_hdl_t | | |
|---|---|---|
| Information about the service or the characteristic that the attribute belongs to. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the GATT Client that accesses to the GATT DataBase. |
| uint8_t | service_id | ID of the service that the attribute belongs to. |
| uint8_t | char_id | ID of the Characteristic that the attribute belongs to. |

### ◆ st_ble_gatts_db_access_evt_t

| struct st_ble_gatts_db_access_evt_t | | |
|---|---|---|
| This structure notifies that the GATT Database has been accessed from a GATT Client. | | |
| Data Fields | | |
| st_ble_gatts_db_conn_hdl_t * | p_handle | Information about the service or the characteristic that the attribute belongs to. |
| st_ble_gatts_db_params_t * | p_params | Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client. |

### ◆ st_ble_gatts_conn_evt_t

| struct st_ble_gatts_conn_evt_t |
|---|

| This structure notifies that the link with the GATT Client has been established. | | |
|---|---|---|
| Data Fields | | |
| st_ble_dev_addr_t * | p_addr | Address of the GATT Client. |

◆ **st_ble_gatts_disconn_evt_t**

| struct st_ble_gatts_disconn_evt_t | | |
|---|---|---|
| This structure notifies that the link with the GATT Client has been disconnected. | | |
| Data Fields | | |
| st_ble_dev_addr_t * | p_addr | Address of the GATT Client. |

◆ **st_ble_gatts_ex_mtu_req_evt_t**

| struct st_ble_gatts_ex_mtu_req_evt_t | | |
|---|---|---|
| This structure notifies that a MTU Exchange Request PDU has been received from a GATT Client. | | |
| Data Fields | | |
| uint16_t | mtu | Maximum receive MTU size by GATT Client. |

◆ **st_ble_gatts_cfm_evt_t**

| struct st_ble_gatts_cfm_evt_t | | |
|---|---|---|
| This structure notifies that a Confirmation PDU has been received from a GATT Client. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic sent by the Indication PDU. |

◆ **st_ble_gatts_read_by_type_rsp_evt_t**

| struct st_ble_gatts_read_by_type_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Read By Type Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic read by the Read By Type Request PDU. |

◆ **st_ble_gatts_read_rsp_evt_t**

| struct st_ble_gatts_read_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Read Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic read by the Read Request PDU. |

### ◆ st_ble_gatts_read_blob_rsp_evt_t

| struct st_ble_gatts_read_blob_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Read Blob Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic read by the Read Blob Request PDU. |

### ◆ st_ble_gatts_read_multi_rsp_evt_t

| struct st_ble_gatts_read_multi_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Read Multiple Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint8_t | count | The number of attribute read by the Read Multiple Request PDU. |
| uint16_t * | p_attr_hdl_list | The list of attribute read by the Read Multiple Request PDU. |

### ◆ st_ble_gatts_write_rsp_evt_t

| struct st_ble_gatts_write_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Write Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic written by the Write Request PDU. |

### ◆ st_ble_gatts_prepare_write_rsp_evt_t

| struct st_ble_gatts_prepare_write_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Prepare Write Response PDU has been sent from GATT Server. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute handle identifying the Characteristic written by the Prepare Write Request PDU. |
| uint16_t | length | The length of written bytes by the Prepare Write Request PDU. |
| uint16_t | offset | The offset of the first octet to be written. |

### ◆ st_ble_gatts_exe_write_rsp_evt_t

| struct st_ble_gatts_exe_write_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Execute Write Response PDU has been sent from GATT Server. | | |
| Data Fields | | |

| | | |
|---|---|---|
| uint8_t | exe_flag | The flag that indicates whether execution or cancellation.<br>value: 0x00 — Cancellation.<br>value: 0x01 — Execution. |

### ◆ st_ble_gatts_db_uuid_cfg_t

struct st_ble_gatts_db_uuid_cfg_t

A structure that defines the information on the position where UUIDs are used.

Data Fields

| | | |
|---|---|---|
| uint16_t | offset | The position of the defined UUID is specified by offset value in uuid_table of st_ble_gatts_db_cfg_t. |
| uint16_t | first | The attribute handle that indicates the first position in st_ble_gatts_db_attr_cfg_t for the defined UUID is specified. |
| uint16_t | last | The attribute handle that indicates the last position in st_ble_gatts_db_attr_cfg_t for the defined UUID is specified. |

### ◆ st_ble_gatts_db_attr_cfg_t

struct st_ble_gatts_db_attr_cfg_t

A structure that defines the detailed information of the attributes.

Data Fields

| | | |
|---|---|---|
| uint8_t | desc_prop | The properties of attribute are specified.<br><br>Set the following properties by a bitwise OR.<br>macro: BLE_GATT_DB_READ(0x01) — Allow clients to read.<br>macro: BLE_GATT_DB_WRITE(0x02) — Allow clients to write.<br>macro: BLE_GATT_DB_WRITE_WITHOUT_RSP(0x04) — Allow clients to write.<br>macro: BLE_GATT_DB — Allow clients |

<table>
<tr><td></td><td></td><td>_READ_WRITE (0x07)</td><td>to access of all.</td></tr>
<tr><td>uint8_t</td><td>aux_prop</td><td colspan="2">The auxiliary properties of attribute are specified.<br><br>Set the following properties by a bitwise OR.<br><br><table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATT_DB_NO_AUXILIARY_PROPERTY(0x00)</td><td>No auxiliary properties. It is invalid when used with other properties at the same time.</td></tr><tr><td>BLE_GATT_DB_FIXED_LENGTH_PROPERTY(0x01)</td><td>Fixed length attribute value.</td></tr><tr><td>BLE_GATT_DB_AUTHORIZATION_PROPERTY(0x02)</td><td>Attributes requiring authorization.</td></tr><tr><td>BLE_GATT_DB_ATTR_DISABLED(0x10)</td><td>The attribute is disabled. If this value is set, the attribute cannot be found and accessed by a GATT Client. It is invalid when used with other properties at the same time.</td></tr><tr><td>BLE_GATT_DB_128_BIT_UUID_FORMAT(0x20)</td><td>Attribute with 128 bit UUID. If this macro is not set, the attribute value is 16-bits UUID.</td></tr><tr><td>BLE_GATT_DB_PEER_SPECIFIC_VAL_PROPE</td><td>Attribute managed by each GATT</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| | | RTY(0x40) Client.<br>BLE_GATT_DB_CONST_ATTR_VAL_PROPERTY(0x80) Fixed attribute value. Writing from Client and setting from Server are prohibited. |
| uint16_t | length | The length of the attribute value is specified. |
| uint16_t | next | The position of the next attribute with the same UUID as the defined attribute is specified by an attribute handle. |
| uint16_t | uuid_offset | The storage area of attribute value.<br><br>UUID of the defined attribute is set by specifying the position of the UUID registered in uuid_table of st_ble_gatts_db_cfg_t with the array offset value. |
| uint8_t * | p_data_offset | Storage area of attribute value.<br><br>The address in the array registered in No.1-No.4 is specified to set the attribute value storage area of the defined attribute. |

### ◆ st_ble_gatts_db_attr_list_t

| struct st_ble_gatts_db_attr_list_t | | |
|---|---|---|
| The number of attributes are stored. | | |
| Data Fields | | |
| uint8_t | count | The number of the services or the characteristics. |

### ◆ st_ble_gatts_db_char_cfg_t

| struct st_ble_gatts_db_char_cfg_t | | |
|---|---|---|
| A structure that defines the detailed information of the characteristics. | | |
| Data Fields | | |
| st_ble_gatts_db_attr_list_t | list | The total number of attributes in the defined characteristic is |

| | | |
|---|---|---|
| | | specified. |
| uint16_t | start_hdl | The first attribute handle of the characteristic is specified. |
| uint8_t | service_id | The index of service to which the characteristic belongs is specified. |

### ◆ st_ble_gatts_db_serv_cfg_t

struct st_ble_gatts_db_serv_cfg_t

A structure that defines the detailed information of the characteristics.

Data Fields

| | | |
|---|---|---|
| st_ble_gatts_db_attr_list_t | list | The total number of service declarations in the defined service is specified. |
| uint32_t | desc | The properties of the defined service are specified.<br><br>Set the security level, the security mode and the key size with a bitwise OR. The bit0-bit3 are specified as the security level. Select one of the following.<br><br>macro / description<br>BLE_GATT_DB_SER_SECURITY_UNAUTH(0x00000001) — Unauthenticated pairing(Security Mode1 Security Level 2, Security Mode 2 Security Level 1) Unauthenticated pairing is required to access the service.<br>BLE_GATT_DB_SER_SECURITY_AUTH(0x00000002) — Authenticated pairing(Security Mode1 Security Level 3, Security Mode 2 Security Level 2) Authenticated pairing is required to |

| | | | |
|---|---|---|---|
| | | | access the service. |
| | | | BLE_GATT_DB_SER_SECURITY_SECONN(0x00000004) Authenticated LE secure connections that generates 16bytes LTK(Security Mode1 Security Level 4) Authenticated LE secure connections pairing that generates 16bytes LTK is required to access the service. If this bit is set, bit24-27 are ignored. |
| | | | The bit4 is specified as the security mode.<br><br>macro: description<br>BLE_GATT_DB_SER_SECURITY_ENC(0x00000010): Encryption Encryption by the LTK exchanged in pairing is required to access.<br><br>If the security requirement of the service is not needed, specify the bit0-bit4 to BLE_GATT_DB_SER_NO_SECURITY_PROPERTY(0x00000000) .(Security Mode1 Security Level 1)<br>The bit24-bit27 are specified as the key size required by the defined service.<br>Select one of the following.<br><br>macro: description<br>BLE_GATT_DB_SER_ENCRYP: 7-byte encryption |

| | | | |
|---|---|---|---|
| | | T_KEY_SIZE_7(0x01000000) | key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_8(0x02000000) | 8-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_9(0x03000000) | 9-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_10(0x04000000) | 10-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_11(0x05000000) | 11-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_12(0x06000000) | 12-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_13(0x07000000) | 13-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_14(0x08000000) | 14-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_15(0x09000000) | 15-byte encryption key. |
| | | BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_16(0x0A000000) | 16-byte encryption key. |
| | | BLE_GATT_DB_SER_ENC_KEY_SIZE_DONT_CARE(0x0000 | 7-byte or larger encryption key. |

| | | |
|---|---|---|
| | | 0000)<br>Other bits are reserved. |
| uint16_t | start_hdl | The start attribute handle of the defined service is specified. |
| uint16_t | end_hdl | The end attribute handle of the defined service is specified. |
| uint8_t | char_start_idx | The start index of the characteristic that belongs to the defined service is specified. |
| uint8_t | char_end_idx | The end index of the characteristic that belongs to the defined service is specified. |

### ◆ st_ble_gatts_db_cfg_t

| struct st_ble_gatts_db_cfg_t | | |
|---|---|---|
| This is the structure of GATT Database that is specified in R_BLE_GATTS_SetDbInst(). | | |
| Data Fields | | |
| const uint8_t * | p_uuid_table | The array to register the UUID to be used. |
| uint8_t * | p_attr_val_table | The array to register variable attribute values. |
| const uint8_t * | p_const_attr_val_table | The array to register fixed attribute values. |
| uint8_t * | p_rem_spec_val_table | The array to manage the attribute values handled for each GATT client. |
| const uint8_t * | p_const_rem_spec_val_table | The array to register the default of the attribute value handled by each GATT client. |
| const st_ble_gatts_db_uuid_cfg_t * | p_uuid_cfg | The array to register information on the position where UUIDs are used. |
| const st_ble_gatts_db_attr_cfg_t * | p_attr_cfg | The array to register the detailed information of attributes. |
| const st_ble_gatts_db_char_cfg_t * | p_char_cfg | The array to register the detailed information of characteristics. |
| const st_ble_gatts_db_serv_cfg_t * | p_serv_cfg | The array to register the detailed information of services. |
| uint8_t | serv_cnt | The number of services included in the GATT Database. |

| | | |
|---|---|---|
| uint8_t | char_cnt | The number of characteristics included in the GATT Database. |
| uint8_t | uuid_type_cnt | The number of UUIDs included in the GATT Database. |
| uint8_t | peer_spec_val_cnt | The total size of attribute value that needs to be managed for each GATT client. |

### ◆ st_ble_gatts_evt_data_t

| struct st_ble_gatts_evt_data_t | | |
|---|---|---|
| st_ble_gatts_evt_data_t is the type of the data notified in a GATT Server Event. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the GATT Client. |
| uint16_t | param_len | The size of GATT Server Event parameters. |
| void * | p_param | GATT Server Event parameters. This parameter differs in each GATT Server Event. |

## Typedef Documentation

### ◆ ble_gatts_app_cb_t

ble_gatts_app_cb_t

ble_gatts_app_cb_t is the GATT Server Event callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | event_type | The type of GATT Server Event. |
| [in] | event_result | The result of GATT Server Event |
| [in] | p_event_data | Data notified by GATT Server Event. |

**Returns**

none

## Enumeration Type Documentation

### ◆ e_r_ble_gatts_evt_t

| enum e_r_ble_gatts_evt_t | |
|---|---|
| GATT Server Event Identifier. | |
| Enumerator | |
| BLE_GATTS_EVENT_EX_MTU_REQ | MTU Exchange Request has been received.<br>This event notifies the application layer that a MTU Exchange Request PDU has been received from a GATT Client. Need to reply to the request by R_BLE_GATTS_RspExMtu().<br>**Event Code: 0x3002**<br>**Event Data:**<br>st_ble_gatts_ex_mtu_req_evt_tBLE_GATTS_EVENT_EX_MTU_REQ |
| BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP | Read By Type Response has been sent.<br>This event notifies the application layer that a Read By Type Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x3009**<br>**Event Data:**<br>st_ble_gatts_read_by_type_rsp_evt_tBLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP |
| BLE_GATTS_EVENT_READ_RSP_COMP | Read Response has been sent.<br>This event notifies the application layer that a Read Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x300B**<br>**Event Data:**<br>st_ble_gatts_read_rsp_evt_tBLE_GATTS_EVENT_READ_RSP_COMP |
| BLE_GATTS_EVENT_READ_BLOB_RSP_COMP | Read Blob Response has been sent.<br>This event notifies the application layer that a Read Blob Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x300D**<br>**Event Data:** |

| | |
|---|---|
| | st_ble_gatts_read_blob_rsp_evt_tBLE_GATTS_EVENT_READ_BLOB_RSP_COMP |
| BLE_GATTS_EVENT_READ_MULTI_RSP_COMP | Read Multiple Response has been sent.<br>This event notifies the application layer that a Read Multiple Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x300F**<br>**Event Data:**<br>st_ble_gatts_read_multi_rsp_evt_tBLE_GATTS_EVENT_READ_MULTI_RSP_COMP |
| BLE_GATTS_EVENT_WRITE_RSP_COMP | Write Response has been sent.<br>This event notifies the application layer that a Write Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x3013**<br>**Event Data:**<br>st_ble_gatts_write_rsp_evt_tBLE_GATTS_EVENT_WRITE_RSP_COMP |
| BLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP | Prepare Write Response has been sent.<br>This event notifies the application layer that a Prepare Write Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x3017**<br>**Event Data:**<br>st_ble_gatts_prepare_write_rsp_evt_tBLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP |
| BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP | Execute Write Response has been sent.<br>This event notifies the application layer that a Execute Write Response PDU has been sent from GATT Server to the GATT Client.<br>**Event Code: 0x3019**<br>**Event Data:**<br>st_ble_gatts_exe_write_rsp_evt_tBLE_GATTS_EVENT_EXE_WRITE_RSP_COMP |
| BLE_GATTS_EVENT_HDL_VAL_CNF | Confirmation has been received.<br>This event notifies the application layer that a |

| | |
|---|---|
| | Confirmation PDU has been received from a GATT Client.<br>**Event Code: 0x301E**<br>**Event Data:**<br>st_ble_gatts_cfm_evt_tBLE_GATTS_EVENT_HDL_VAL_CNF |
| BLE_GATTS_EVENT_DB_ACCESS_IND | The GATT Database has been accessed from a GATT Client.<br>This event notifies the application layer that the GATT Database has been accessed from a GATT Client.<br>**Event Code: 0x3040**<br>**Event Data:**<br>st_ble_gatts_db_access_evt_tBLE_GATTS_EVENT_DB_ACCESS_IND |
| BLE_GATTS_EVENT_CONN_IND | A connection has been established.<br>This event notifies the application layer that the link with the GATT Client has been established.<br>**Event Code: 0x3081**<br>**Event Data:**<br>st_ble_gatts_conn_evt_tBLE_GATTS_EVENT_CONN_IND |
| BLE_GATTS_EVENT_DISCONN_IND | A connection has been disconnected.<br>This event notifies the application layer that the link with the GATT Client has been disconnected.<br>**Event Code: 0x3082**<br>**Event Data:**<br>st_ble_gatts_disconn_evt_tBLE_GATTS_EVENT_DISCONN_IND |
| BLE_GATTS_EVENT_INVALID | Invalid GATT Server Event.<br>**Event Code: 0x30FF**<br>**Event Data:**<br>noneBLE_GATTS_EVENT_INVALID |

## Function Documentation

### ◆ R_BLE_GATTS_Init()

ble_status_t R_BLE_GATTS_Init ( uint8_t *cb_num*)

This function initializes the GATT Server and registers the number of the callbacks for GATT Server event.

Specify the cb_num parameter to a value between 1 and BLE_GATTS_MAX_CB.
R_BLE_GATTS_RegisterCb() registers the callback.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb_num | The number of callbacks to be registered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The cb_num parameter is out of range. |

### ◆ R_BLE_GATTS_SetDbInst()

ble_status_t R_BLE_GATTS_SetDbInst ( st_ble_gatts_db_cfg_t * *p_db_inst*)

This function sets GATT Database to host stack.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_db_inst | GATT Database to be set. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows.<br>• The db_inst parameter is specified as NULL.<br>• The array in the db_inst is specified as NULL. |

### ◆ R_BLE_GATTS_RegisterCb()

ble_status_t R_BLE_GATTS_RegisterCb ( ble_gatts_app_cb_t *cb*, uint8_t *priority* )

This function registers a callback for GATT Server event.

The number of the callback that may be registered by this function is the value specified by R_BLE_GATTS_Init().
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb | Callback function for GATT Server event. |
| [in] | priority | The priority of the callback function.<br>Valid range is 1 <= priority <= BLE_GATTS_MAX_CB.<br>A lower priority number means a higher priority level. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The cb parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The priority parameter is out of range. |
| BLE_ERR_CONTEXT_FULL(0x000B) | Host stack has already registered the maximum number of callbacks. |

### ◆ R_BLE_GATTS_DeregisterCb()

ble_status_t R_BLE_GATTS_DeregisterCb ( ble_gatts_app_cb_t *cb*)

This function deregisters the callback function for GATT Server event.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb | The callback function to be deregistered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The cb parameter is specified as NULL. |
| BLE_ERR_NOT_FOUND(0x000D) | The callback has not been registered. |

### ◆ R_BLE_GATTS_Notification()

ble_status_t R_BLE_GATTS_Notification ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_ntf_data* )

This function sends a notification of an attribute's value.

The maximum length of the attribute value that can be sent with notification is MTU-3.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to be sent the notification. |
| [in] | p_ntf_data | The attribute value to send. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_ntf_data parameter or the value field in the value field in the p_ntf_data parameter is NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The value_len field in the value field in the p_ntf_data parameter is 0 or the attr_hdl field in the p_ntf_data parameters is 0. |
| BLE_ERR_INVALID_OPERATION(0x0009) | This function was called while processing other request. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl was not found. |

### ◆ R_BLE_GATTS_Indication()

ble_status_t R_BLE_GATTS_Indication ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_ind_data* )

This function sends a indication of an attribute's value.

The maximum length of the attribute value that can be sent with indication is MTU-3.
The result of this API call is returned by a return value.
The remote device that receives a indication sends a confirmation.
BLE_GATTS_EVENT_HDL_VAL_CNF event notifies the application layer that the confirmation has been received.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to be sent the indication. |
| [in] | p_ind_data | The attribute value to send. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_ind_data parameter or the value field in the value field in the p_ind_data parameter is NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The value_len field in the value field in the p_ind_data parameter is 0 or the attr_hdl field in the p_ind_data parameters is 0. |
| BLE_ERR_INVALID_OPERATION(0x0009) | This function was called while processing other request. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl was not found. |

### ◆ R_BLE_GATTS_GetAttr()

ble_status_t R_BLE_GATTS_GetAttr ( uint16_t *conn_hdl*, uint16_t *attr_hdl*, st_ble_gatt_value_t * *p_value* )

This function gets a attribute value from the GATT Database.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL. |
| [in] | attr_hdl | The attribute handle of the attribute value to be retrieved. |
| [out] | p_value | The attribute value to be retrieved. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_value parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database. |
| BLE_ERR_INVALID_STATE(0x0008) | The attribute is not in a state to be read. |
| BLE_ERR_INVALID_OPERATION(0x0009) | The attribute cannot be read. |
| BLE_ERR_NOT_FOUND(0x000D) | The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by the conn_hdl parameter was not found. |

### ◆ R_BLE_GATTS_SetAttr()

ble_status_t R_BLE_GATTS_SetAttr ( uint16_t *conn_hdl*, uint16_t *attr_hdl*, st_ble_gatt_value_t * *p_value* )

This function sets an attribute value to the GATT Database.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL. |
| [in] | attr_hdl | The attribute handle of the attribute value to be set. |
| [in] | p_value | The attribute value to be set. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_value parameter is specified as NULL. |
| BLE_ERR_INVALID_DATA(0x0002) | The write size is larger than the length of the attribute value. |
| BLE_ERR_INVALID_ARG(0x0003) | The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database. |
| BLE_ERR_INVALID_STATE(0x0008) | The attribute is not in a state to be written. |
| BLE_ERR_INVALID_OPERATION(0x0009) | The attribute cannot be written. |
| BLE_ERR_NOT_FOUND(0x000D) | The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by the conn_hdl parameter was not found. |

### ◆ R_BLE_GATTS_SendErrRsp()

ble_status_t R_BLE_GATTS_SendErrRsp ( uint16_t *error_code*)

This function sends an error response to a remote device.

The result is returned from the API.
The error code specified in the callback is notified as Error Response to the remote device.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | error_code | The error codes to be notified the client.<br>It is a bitwise OR of GATT Error Group ID : 0x3000 and the following error codes defined in Core Spec and Core Spec Supplement. |

| Error Code | description |
|---|---|
| BLE_ERR_GATT_INVALID_HANDLE(0x3001) | Invalid attribute handle |
| BLE_ERR_GATT_READ_NOT_PERMITTED(0x3002) | The attribute cannot be read. |
| BLE_ERR_GATT_WRITE_NOT_PERMITTED(0x3003) | The attribute cannot be written. |
| BLE_ERR_GATT_INVALID_PDU(0x3004) | Invalid PDU. |
| BLE_ERR_GATT_INSUFFICIENT_AUTHENTICATION(0x3005) | The authentication to access the attribute is insufficient. |
| BLE_ERR_GATT_REQUEST_NOT_SUPPORTED(0x3006) | The request is not supported. |
| BLE_ERR_GATT_INVALID_OFFSET(0x3007) | The specified offset is larger than the length of the attribute value. |
| BLE_ERR_GATT_INSUFFICIENT_AUTHO | Authorization is required to access |

| | | | | | |
|---|---|---|---|---|---|
| | | | | RIZATION(0x3008) | the attribute. |
| | | | | BLE_ERR_GATT_PREPARE_WRITE_QUEUE_FULL(0x3009) | The Write Queue in the GATT Server is full. |
| | | | | BLE_ERR_GATT_ATTRIBUTE_NOT_FOUND(0x300A) | The specified attribute is not found. |
| | | | | BLE_ERR_GATT_ATTRIBUTE_NOT_LONG(0x300B) | The attribute cannot be read by Read Blob Request. |
| | | | | BLE_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE(0x300C) | The Encryption Key Size is insufficient. |
| | | | | BLE_ERR_GATT_INVALID_ATTRIBUTE_LEN(0x300D) | The length of the specified attribute is invalid. |
| | | | | BLE_ERR_GATT_UNLIKELY_ERROR(0x300E) | Because an error has occurred, the process cannot be advanced. |
| | | | | BLE_ERR_GATT_INSUFFICIENT_ENCRYPTION(0x300F) | Encryption is required to access the attribute. |
| | | | | BLE_ERR_GATT_UNSUPPORTED_GROUP_TYPE(0x3010) | The type of the specified attribute is not supported. |
| | | | | BLE_ERR_GATT_INSUFFICIENT_RESOURCES(0x3011) | The resource to complete the request is insufficient. |
| | | | | 0x3080 - | Application |

| | | | | |
|---|---|---|---|---|
| | | | 0x309F | Error. The upper layer defines the error codes. |
| | | | 0x30E0 - 0x30FF | The error code defined in Common Profile and Service Error Core Specification Supplement( CSS). CSS ver.7 defines the error codes from 0x30FC to 0x30FF. |
| | | | BLE_ERR_GATT_WRITE_REQ_REJECTED(0x30FC) | The Write Request has not been completed due to the reason other than Permission. |
| | | | BLE_ERR_GATT_CCCD_IMPROPERLY_CFG(0x30FD) | The CCCD is set to be invalid. |
| | | | BLE_ERR_GATT_PROC_ALREADY_IN_PROGRESS(0x30FE) | The request is now in progress. |
| | | | BLE_ERR_GATT_OUT_OF_RANGE(0x30FF) | The attribute value is out of range. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The Group ID of the error_code parameter is not 0x3000, or it is 0x3000. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other error response,this function was called. |

### ◆ R_BLE_GATTS_RspExMtu()

ble_status_t R_BLE_GATTS_RspExMtu ( uint16_t *conn_hdl*, uint16_t *mtu* )

This function replies to a MTU Exchange Request from a remote device.

BLE_GATTS_EVENT_EX_MTU_REQ event notifies the application layer that a MTU Exchange Request has been received. Therefore when the callback has received the event, call this function.
The new MTU is the minimum of the mtu parameter specified by this function and the mtu field in BLE_GATTS_EVENT_EX_MTU_REQ event.
Default MTU size is 23 bytes.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to be sent MTU Exchange Response. |
| [in] | mtu | The maximum size(in bytes) of the GATT PDU that GATT Server can receive.<br>Valid range is 23 <= mtu <= 247. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The mtu parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | This function was called while processing other request. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl was not found. |

### ◆ R_BLE_GATTS_SetPrepareQueue()

ble_status_t R_BLE_GATTS_SetPrepareQueue ( st_ble_gatt_pre_queue_t * *p_pre_queues*, uint8_t *queue_num* )

Register prepare queue and buffer in Host Stack.

This function registers the prepare queue and buffer for long chracteristic write and reliable writes. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_pre_queues | The prepare write queues to be registered. |
| [in] | queue_num | The number of prepare write queues to be registered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_pre_queue parameter is specified as NULL. |

## GATT_CLIENT

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_GATTC_Init (uint8_t cb_num)<br>This function initializes the GATT Client and registers the number of the callbacks for GATT Client event. More... |
| ble_status_t | R_BLE_GATTC_RegisterCb (ble_gattc_app_cb_t cb, uint8_t priority)<br>This function registers a callback function for GATT Client event. More... |
| ble_status_t | R_BLE_GATTC_DeregisterCb (ble_gattc_app_cb_t cb)<br>This function deregisters the callback function for GATT Client event. More... |
| ble_status_t | R_BLE_GATTC_ReqExMtu (uint16_t conn_hdl, uint16_t mtu)<br>This function sends a MTU Exchange Request PDU to a GATT Server |

| | |
|---|---|
| | in order to change the current MTU. More... |
| ble_status_t | R_BLE_GATTC_DiscAllPrimServ (uint16_t conn_hdl)<br>This function discovers all Primary Services in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscPrimServ (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type)<br>This function discovers Primary Service specified by p_uuid in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscAllSecondServ (uint16_t conn_hdl)<br>This function discovers all Secondary Services in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscIncServ (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range)<br>This function discovers Included Services within the specified attribute handle range in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscAllChar (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range)<br>This function discovers Characteristic within the specified attribute handle range in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscCharByUuid (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type, st_ble_gatt_hdl_range_t *p_range)<br>This function discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_DiscAllCharDesc (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range)<br>This function discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_ReadChar (uint16_t conn_hdl, uint16_t value_hdl)<br>This function reads a Characteristic/Characteristic Descriptor in a GATT Server. More... |
| | |

| | |
|---|---|
| ble_status_t | R_BLE_GATTC_ReadCharUsingUuid (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type, st_ble_gatt_hdl_range_t *p_range)<br>This function reads a Characteristic in a GATT Server using a specified UUID. More... |
| ble_status_t | R_BLE_GATTC_ReadLongChar (uint16_t conn_hdl, uint16_t value_hdl, uint16_t offset)<br>This function reads a Long Characteristic in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_ReadMultiChar (uint16_t conn_hdl, st_ble_gattc_rd_multi_req_param_t *p_list)<br>This function reads multiple Characteristics in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_WriteCharWithoutRsp (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data)<br>This function writes a Characteristic in a GATT Server without response. More... |
| ble_status_t | R_BLE_GATTC_SignedWriteChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data)<br>This function writes Signed Data to a Characteristic in a GATT Server without response. More... |
| ble_status_t | R_BLE_GATTC_WriteChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data)<br>This function writes a Characteristic in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_WriteLongChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data, uint16_t offset)<br>This function writes a Long Characteristic in a GATT Server. More... |
| ble_status_t | R_BLE_GATTC_ReliableWrites (uint16_t conn_hdl, st_ble_gattc_reliable_writes_char_pair_t *p_char_pair, uint8_t pair_num, uint8_t auto_flag)<br>This function performs the Reliable Writes procedure described in GATT Specification. More... |
| ble_status_t | R_BLE_GATTC_ExecWrite (uint16_t conn_hdl, uint8_t exe_flag)<br>If the auto execute of Reliable Writes is not specified by |

| | |
|---|---|
| | R_BLE_GATTC_ReliableWrites(), this function is used to execute a write to Characteristic. More... |

## Detailed Description

### Data Structures

| | |
|---|---|
| struct | st_ble_gatt_hdl_range_t<br>Attribute handle range. More... |
| struct | st_ble_gattc_reliable_writes_char_pair_t<br>This is used in R_BLE_GATTC_ReliableWrites() to specify the pair of Characteristic Value and Characteristic Value Handle. More... |
| struct | st_ble_gattc_conn_evt_t<br>This structure notifies that the link with the GATT Server has been established. More... |
| struct | st_ble_gattc_disconn_evt_t<br>This structure notifies that the link with the GATT Server has been disconnected. More... |
| struct | st_ble_gattc_ex_mtu_rsp_evt_t<br>This structure notifies that a MTU Exchange Response PDU has been received from a GATT Server. More... |
| struct | st_ble_gattc_serv_16_evt_t<br>This structure notifies that a 16-bit UUID Service has been discovered. More... |
| struct | st_ble_gattc_serv_128_evt_t<br>This structure notifies that a 128-bit UUID Service has been discovered. More... |
| struct | st_ble_gattc_inc_serv_16_evt_t<br>This structure notifies that a 16-bit UUID Included Service has been discovered. More... |

| | |
|---|---|
| struct | st_ble_gattc_inc_serv_128_evt_t<br>This structure notifies that a 128-bit UUID Included Service has been discovered. More... |
| struct | st_ble_gattc_char_16_evt_t<br>This structure notifies that a 16-bit UUID Characteristic has been discovered. More... |
| struct | st_ble_gattc_char_128_evt_t<br>This structure notifies that a 128-bit UUID Characteristic has been discovered. More... |
| struct | st_ble_gattc_char_desc_16_evt_t<br>This structure notifies that a 16-bit UUID Characteristic Descriptor has been discovered. More... |
| struct | st_ble_gattc_char_desc_128_evt_t<br>This structure notifies that a 128-bit UUID Characteristic Descriptor has been discovered. More... |
| struct | st_ble_gattc_err_rsp_evt_t<br>This structure notifies that a Error Response PDU has been received from a GATT Server. More... |
| struct | st_ble_gattc_ntf_evt_t<br>This structure notifies that a Notification PDU has been received from a GATT Server. More... |
| struct | st_ble_gattc_ind_evt_t<br>This structure notifies that a Indication PDU has been received from a GATT Server. More... |
| struct | st_ble_gattc_svc_chng_evt_t<br>This structure notifies that a service changed PDU has been received from a GATT Server. More... |
| struct | st_ble_gattc_rd_char_evt_t |

| | |
|---|---|
| | This structure notifies that read response to R_BLE_GATTC_ReadChar() or R_BLE_GATTC_ReadCharUsingUuid() has been received from a GATT Server. More... |
| struct | st_ble_gattc_wr_char_evt_t<br>This structure notifies that write response to R_BLE_GATTC_WriteChar() has been received from a GATT Server. More... |
| struct | st_ble_gattc_att_info_req_ind_evt_t<br>This structure notifies that an att_info_req_ind event has arrived. More... |
| struct | st_ble_gattc_wr_char_without_rsp_evt_t<br>This structure notifies that write response to R_BLE_GATTC_WriteCharWithoutRsp() has been received from a GATT Server. More... |
| struct | st_ble_gattc_rd_multi_char_evt_t<br>This structure notifies that read response to R_BLE_GATTC_ReadMultiChar() has been received from a GATT Server. More... |
| struct | st_ble_gattc_char_part_wr_evt_t<br>This structure notifies that write response to R_BLE_GATTC_WriteLongChar() or R_BLE_GATTC_ReliableWrites() has been received from a GATT Server. More... |
| struct | st_ble_gattc_reliable_writes_comp_evt_t<br>This structure notifies that a response to R_BLE_GATTC_ExecWrite() has been received from a GATT Server. More... |
| struct | st_ble_gattc_rd_multi_req_param_t<br>This is used in R_BLE_GATTC_ReadMultiChar() to specify multiple Characteristics to be read. More... |
| struct | st_ble_gattc_evt_data_t<br>st_ble_gattc_evt_data_t is the type of the data notified in a GATT Client Event. More... |

| | |
|---|---|
| struct | st_ble_gatt_value_t<br>Attribute Value. More... |
| struct | st_ble_gatt_hdl_value_pair_t<br>Attribute handle and attribute Value. More... |

### Macros

| | |
|---|---|
| #define | BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG |
| #define | BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG |
| #define | BLE_GATTC_MAX_CB<br>GATT Client Callback Number. |
| #define | BLE_GATTC_EXEC_AUTO<br>Auto execution. |
| #define | BLE_GATTC_EXEC_NOT_AUTO<br>Not auto execution. |
| #define | BLE_GATTC_RELIABLE_WRITES_MAX_CHAR_PAIR<br>Length of the Queue used with Prepare Write procedure to write a characteristic whose size is larger than MTU. |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_gattc_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data)<br>ble_gattc_app_cb_t is the GATT Client Event callback function type. More... |

### Enumerations

| | |
|---|---|
| enum | e_r_ble_gattc_evt_t<br>GATT Client Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_gatt_hdl_range_t

| struct st_ble_gatt_hdl_range_t | | |
|---|---|---|
| Attribute handle range. | | |
| Data Fields | | |
| uint16_t | start_hdl | Start Attribute Handle. |
| uint16_t | end_hdl | End Attribute Handle. |

### ◆ st_ble_gattc_reliable_writes_char_pair_t

| struct st_ble_gattc_reliable_writes_char_pair_t | | |
|---|---|---|
| This is used in R_BLE_GATTC_ReliableWrites() to specify the pair of Characteristic Value and Characteristic Value Handle. | | |
| Data Fields | | |
| st_ble_gatt_hdl_value_pair_t | write_data | Pair of Characteristic Value and Characteristic Value Handle. |
| uint16_t | offset | Offset that indicates the location to be written.<br><br>Normally, set 0 to this parameter.<br>If this parameter sets to a value other than 0,Adjust the offset parameter and the length of the value to be written not to exceed the length of the Characteristic. |

### ◆ st_ble_gattc_conn_evt_t

| struct st_ble_gattc_conn_evt_t | | |
|---|---|---|
| This structure notifies that the link with the GATT Server has been established. | | |
| Data Fields | | |
| st_ble_dev_addr_t * | p_addr | Address of the GATT Server. |

### ◆ st_ble_gattc_disconn_evt_t

| struct st_ble_gattc_disconn_evt_t | | |
|---|---|---|
| This structure notifies that the link with the GATT Server has been disconnected. | | |
| Data Fields | | |
| st_ble_dev_addr_t * | p_addr | Address of the GATT Server. |

### ◆ st_ble_gattc_ex_mtu_rsp_evt_t

| struct st_ble_gattc_ex_mtu_rsp_evt_t |
|---|
| This structure notifies that a MTU Exchange Response PDU has been received from a GATT Server. |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | mtu | MTU size(in bytes) that GATT Server can receive. |

### ◆ st_ble_gattc_serv_16_evt_t

| struct st_ble_gattc_serv_16_evt_t | | |
|---|---|---|
| This structure notifies that a 16-bit UUID Service has been discovered. | | |
| Data Fields | | |
| st_ble_gatt_hdl_range_t | range | Attribute handle range of the 16-bit UUID service. |
| uint16_t | uuid_16 | Service UUID. |

### ◆ st_ble_gattc_serv_128_evt_t

| struct st_ble_gattc_serv_128_evt_t | | |
|---|---|---|
| This structure notifies that a 128-bit UUID Service has been discovered. | | |
| Data Fields | | |
| st_ble_gatt_hdl_range_t | range | Attribute handle range of the 128-bit UUID service. |
| uint8_t | uuid_128[ BLE_GATT_128_BIT_UUID_SIZE] | Service UUID. |

### ◆ st_ble_gattc_inc_serv_16_evt_t

| struct st_ble_gattc_inc_serv_16_evt_t | | |
|---|---|---|
| This structure notifies that a 16-bit UUID Included Service has been discovered. | | |
| Data Fields | | |
| uint16_t | decl_hdl | Service Declaration handle of the 16-bit UUID Included Service. |
| st_ble_gattc_serv_16_evt_t | service | The contents of the Included Service. |

### ◆ st_ble_gattc_inc_serv_128_evt_t

| struct st_ble_gattc_inc_serv_128_evt_t | | |
|---|---|---|
| This structure notifies that a 128-bit UUID Included Service has been discovered. | | |
| Data Fields | | |
| uint16_t | decl_hdl | Service Declaration handle of the 128-bit UUID Included Service. |
| st_ble_gattc_serv_128_evt_t | service | The contents of the Included Service. |

### ◆ st_ble_gattc_char_16_evt_t

| struct st_ble_gattc_char_16_evt_t | | |
|---|---|---|
| This structure notifies that a 16-bit UUID Characteristic has been discovered. | | |
| Data Fields | | |
| uint16_t | decl_hdl | Attribute handle of Characteristic Declaration. |
| uint8_t | cproperty | Characteristic Properties.<br><br>It is a bitwise OR of the following values.<br>Refer to Core Spec [Vol.3] Generic Attribute Profile(GATT) "3.3.1.1 Characteristic Properties" regarding the details of the Characteristic Properties.<br><br>value / description:<br>0x01 Broadcast property<br>0x02 Read property<br>0x04 Write Without Response property<br>0x08 Write property<br>0x10 Notify property<br>0x20 Indicate property<br>0x40 Authenticated Signed Writes property<br>0x80 Extended Properties property |
| uint16_t | value_hdl | Value Handle of the Characteristic. |
| uint16_t | uuid_16 | Characteristic UUID. |

◆ **st_ble_gattc_char_128_evt_t**

| struct st_ble_gattc_char_128_evt_t |
|---|
| This structure notifies that a 128-bit UUID Characteristic has been discovered. |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | decl_hdl | Attribute Handle of Characteristic Declaration. |
| uint8_t | cproperty | Characteristic Properties.<br><br>It is a bitwise OR of the following values.<br>Refer to Core Spec [Vol.3] Generic Attribute Profile(GATT) "3.3.1.1 Characteristic Properties" regarding the details of the Characteristic Properties.<br><br>value / description:<br>0x01 Broadcast property<br>0x02 Read property<br>0x04 Write Without Response property<br>0x08 Write property<br>0x10 Notify property<br>0x20 Indicate property<br>0x40 Authenticated Signed Writes property<br>0x80 Extended Properties property |
| uint16_t | value_hdl | Value Handle of the Characteristic. |
| uint8_t | uuid_128[ BLE_GATT_128_BIT_UUID_SIZE] | Characteristic UUID. |

### ◆ st_ble_gattc_char_desc_16_evt_t

| struct st_ble_gattc_char_desc_16_evt_t | | |
|---|---|---|
| This structure notifies that a 16-bit UUID Characteristic Descriptor has been discovered. | | |
| Data Fields | | |
| uint16_t | desc_hdl | Attribute Handle of Characteristic Descriptor. |
| uint16_t | uuid_16 | Characteristic Descriptor UUID. |

### ◆ st_ble_gattc_char_desc_128_evt_t

| struct st_ble_gattc_char_desc_128_evt_t | | |
|---|---|---|
| This structure notifies that a 128-bit UUID Characteristic Descriptor has been discovered. | | |
| Data Fields | | |
| uint16_t | desc_hdl | Attribute Handle of Characteristic Descriptor. |
| uint8_t | uuid_128[ BLE_GATT_128_BIT_UUID_SIZE] | Characteristic Descriptor UUID. |

### ◆ st_ble_gattc_err_rsp_evt_t

| struct st_ble_gattc_err_rsp_evt_t | | |
|---|---|---|
| This structure notifies that a Error Response PDU has been received from a GATT Server. | | |
| Data Fields | | |
| uint8_t | op_code | The op code of the ATT Request that causes the Error Response.<br>op_code<br>Exchange MTU Request(0x02)<br>Find Information Request(0x04)<br>Find By Type Value Request(0x06)<br>Read By Type Request(0x08)<br>Read Request(0x0A)<br>Read Blob Request(0x0C)<br>Read Multiple Request(0x0E)<br>Read by Group Type Request(0x10)<br>Write Request(0x12)<br>Prepare Write Request(0x16)<br>Execute Write Request(0x18) |
| uint16_t | attr_hdl | Attribute handle that is target for the request. |
| uint16_t | rsp_code | The error codes notified from the GATT Server.<br><br>It is a bitwise OR of GATT Error Group ID : 0x3000 and the following error codes defined in Core Spec and Core Spec Supplement. |

| Error Code | description |
|---|---|
| BLE_ERR_GATT_INVALID_HANDLE(0x3001) | Invalid attribute handle |
| BLE_ERR_GATT_READ_NOT_PERMITTED(0x3002) | The attribute cannot be read. |
| BLE_ERR_GATT_WRITE_NOT_PERMITTED(0x3003) | The attribute cannot be written. |
| BLE_ERR_GATT_INVALID_PDU(0x3004) | Invalid PDU. |
| BLE_ERR_GATT_INSUFFICIENT_AUTHENTICATION(0x3005) | The authentication to access the attribute is insufficient. |
| BLE_ERR_GATT_REQUEST_NOT_SUPPORTED(0x3006) | The request is not supported. |
| BLE_ERR_GATT_INVALID_OFFSET(0x3007) | The specified offset is larger than the length of the attribute value. |
| BLE_ERR_GATT_INSUFFICIENT_AUTHORIZATION(0x3008) | Authorization is required to access the attribute. |
| BLE_ERR_GATT_PREPARE_WRITE_QUEUE_FULL(0x3009) | The Write Queue in the GATT Server is full. |
| BLE_ERR_GATT_ATTRIBUTE_NOT_FOUND(0x300A) | The specified attribute is not found. |
| BLE_ERR_GATT_ATTRIBUTE_NOT_LONG(0x300B) | The attribute cannot be read by Read Blob Request. |

| | | | | |
|---|---|---|---|---|
| | | | BLE_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE(0x300C) | The Encryption Key Size is insufficient. |
| | | | BLE_ERR_GATT_INVALID_ATTRIBUTE_LEN(0x300D) | The length of the specified attribute is invalid. |
| | | | BLE_ERR_GATT_UNLIKELY_ERROR(0x300E) | Because an error has occurred, the process cannot be advanced. |
| | | | BLE_ERR_GATT_INSUFFICIENT_ENCRYPTION(0x300F) | Encryption is required to access the attribute. |
| | | | BLE_ERR_GATT_UNSUPPORTED_GROUP_TYPE(0x3010) | The type of the specified attribute is not supported. |
| | | | BLE_ERR_GATT_INSUFFICIENT_RESOURCES(0x3011) | The resource to complete the request is insufficient. |
| | | | 0x3080 - 0x309F | Application Error. The upper layer defines the error codes. |
| | | | 0x30E0 - 0x30FF | The error code defined in Common Profile and Service Error Core Specification Supplement(CSS).<br>CSS ver.7 defines the error codes from 0x30FC to 0x30FF. |
| | | | BLE_ERR_GATT_WRITE_REQ_REJECTED(0x30FC) | The Write Request has not been completed |

| | | | |
|---|---|---|---|
| | | | due to the reason other than Permission. |
| | | BLE_ERR_GATT_CCCD_IMPROPERLY_CFG(0x30FD) | The CCCD is set to be invalid. |
| | | BLE_ERR_GATT_PROC_ALREADY_IN_PROGRESS(0x30FE) | The request is now in progress. |
| | | BLE_ERR_GATT_OUT_OF_RANGE(0x30FF) | The attribute value is out of range. |

◆ **st_ble_gattc_ntf_evt_t**

| struct st_ble_gattc_ntf_evt_t | | |
|---|---|---|
| This structure notifies that a Notification PDU has been received from a GATT Server. | | |
| Data Fields | | |
| st_ble_gatt_hdl_value_pair_t | data | Characteristic that causes the Notification. |

◆ **st_ble_gattc_ind_evt_t**

| struct st_ble_gattc_ind_evt_t | | |
|---|---|---|
| This structure notifies that a Indication PDU has been received from a GATT Server. | | |
| Data Fields | | |
| st_ble_gatt_hdl_value_pair_t | data | Characteristic that causes the Indication. |

◆ **st_ble_gattc_svc_chng_evt_t**

| struct st_ble_gattc_svc_chng_evt_t | | |
|---|---|---|
| This structure notifies that a service changed PDU has been received from a GATT Server. | | |
| Data Fields | | |
| uint16_t | ind_cfg | Client Characteristic Configuration Descriptor value. |

◆ **st_ble_gattc_rd_char_evt_t**

| struct st_ble_gattc_rd_char_evt_t | | |
|---|---|---|
| This structure notifies that read response to R_BLE_GATTC_ReadChar() or R_BLE_GATTC_ReadCharUsingUuid() has been received from a GATT Server. | | |
| Data Fields | | |
| | | |

| | | |
|---|---|---|
| st_ble_gatt_hdl_value_pair_t | read_data | The contents of the Characteristic that has been read. |

◆ **st_ble_gattc_wr_char_evt_t**

| struct st_ble_gattc_wr_char_evt_t | | |
|---|---|---|
| This structure notifies that write response to R_BLE_GATTC_WriteChar() has been received from a GATT Server. | | |
| Data Fields | | |
| uint16_t | value_hdl | Value Handle of the Characteristic/Characteristic Descriptor that has been written. |

◆ **st_ble_gattc_att_info_req_ind_evt_t**

| struct st_ble_gattc_att_info_req_ind_evt_t | | |
|---|---|---|
| This structure notifies that an att_info_req_ind event has arrived. | | |
| Data Fields | | |
| uint16_t | value_hdl | Value Handle of the Characteristic/Characteristic Descriptor that information is required. |

◆ **st_ble_gattc_wr_char_without_rsp_evt_t**

| struct st_ble_gattc_wr_char_without_rsp_evt_t | | |
|---|---|---|
| This structure notifies that write response to R_BLE_GATTC_WriteCharWithoutRsp() has been received from a GATT Server. | | |
| Data Fields | | |
| uint16_t | value_hdl | Value Handle of the Characteristic/Characteristic Descriptor that has been written. |

◆ **st_ble_gattc_rd_multi_char_evt_t**

| struct st_ble_gattc_rd_multi_char_evt_t | | |
|---|---|---|
| This structure notifies that read response to R_BLE_GATTC_ReadMultiChar() has been received from a GATT Server. | | |
| Data Fields | | |
| uint16_t | value_hdl_num | The number of Value Handles of the Characteristics that has been read. |
| st_ble_gatt_value_t | multi_char_val | The contents of multiple Characteristics that have been read. |

### ◆ st_ble_gattc_char_part_wr_evt_t

| struct st_ble_gattc_char_part_wr_evt_t | | |
|---|---|---|
| This structure notifies that write response to R_BLE_GATTC_WriteLongChar() or R_BLE_GATTC_ReliableWrites() has been received from a GATT Server. | | |
| Data Fields | | |
| st_ble_gatt_hdl_value_pair_t | write_data | The data to be written to the Characteristic/Long Characteristic/Long Characteristic Descriptor. |
| uint16_t | offset | Offset that indicates the location to be written. |

### ◆ st_ble_gattc_reliable_writes_comp_evt_t

| struct st_ble_gattc_reliable_writes_comp_evt_t | | |
|---|---|---|
| This structure notifies that a response to R_BLE_GATTC_ExecWrite() has been received from a GATT Server. | | |
| Data Fields | | |
| uint8_t | exe_flag | This field indicates the command of the Execute Write that has been done.<br>value: 0x00 — Cancel the write.<br>value: 0x01 — Execute the write. |

### ◆ st_ble_gattc_rd_multi_req_param_t

| struct st_ble_gattc_rd_multi_req_param_t | | |
|---|---|---|
| This is used in R_BLE_GATTC_ReadMultiChar() to specify multiple Characteristics to be read. | | |
| Data Fields | | |
| uint16_t * | p_hdl_list | List of Value Handles that point the Characteristics to be read. |
| uint16_t | list_count | The number of Value Handles included in the hdl_list parameter. |
| uint16_t * | p_len_list | List of Value Length that point the Characteristics to be read. |

### ◆ st_ble_gattc_evt_data_t

| struct st_ble_gattc_evt_data_t |
|---|
| st_ble_gattc_evt_data_t is the type of the data notified in a GATT Client Event. |
| Data Fields |

| | | |
|---|---|---|
| uint16_t | conn_hdl | Connection handle identifying the GATT Server. |
| uint16_t | param_len | The size of GATT Client Event parameters. |
| void * | p_param | GATT Client Event parameters. This parameter differs in each GATT Client Event. |

### ◆ st_ble_gatt_value_t

| struct st_ble_gatt_value_t | | |
|---|---|---|
| Attribute Value. | | |
| Data Fields | | |
| uint16_t | value_len | Length of the attribute value. |
| uint8_t * | p_value | Attribute Value. |

### ◆ st_ble_gatt_hdl_value_pair_t

| struct st_ble_gatt_hdl_value_pair_t | | |
|---|---|---|
| Attribute handle and attribute Value. | | |
| Data Fields | | |
| uint16_t | attr_hdl | Attribute Handle. |
| st_ble_gatt_value_t | value | Attribute Value. |

## Macro Definition Documentation

### ◆ BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG

| #define BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG |
|---|
| GATT Execute Write Cancel Flag. |

### ◆ BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG

| #define BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG |
|---|
| GATT Execute Write Execute Flag. |

## Typedef Documentation

### ◆ ble_gattc_app_cb_t

ble_gattc_app_cb_t

ble_gattc_app_cb_t is the GATT Client Event callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | event_type | The type of GATT Client Event. |
| [in] | event_result | The result of GATT Client Event |
| [in] | p_event_data | Data notified by GATT Client Event. |

**Returns**

none

## Enumeration Type Documentation

### ◆ e_r_ble_gattc_evt_t

enum e_r_ble_gattc_evt_t

GATT Client Event Identifier.

| Enumerator | |
|---|---|
| BLE_GATTC_EVENT_ERROR_RSP | This event notifies the application layer that a problem has occurred in the GATT Server while processing a request from GATT Client.<br>When GATT Client has received a Error Response PDU from a GATT Server, BLE_GATTC_EVENT_ERROR_RSP event is notified the application layer.<br>**Event Code: 0x4001**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_err_rsp_evt_tBLE_GATTC_EVENT_ERROR_RSP |
| BLE_GATTC_EVENT_EX_MTU_RSP | This event notifies the application layer that a MTU Exchange Response PDU has been received from a GATT Server. |

| | |
|---|---|
| | **Event Code: 0x4003**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a Exchange MTU Response since GATT Client sent a Exchange MTU Request PDU to the GATT Server.<br>**Event Data:**<br>st_ble_gattc_ex_mtu_rsp_evt_tBLE_GATTC_EVENT_EX_MTU_RSP |
| BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP | When the read of Characteristic specified by UUID has been completed, this event is notified to the application layer.<br>**Event Code: 0x4009**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a Exchange MTU Response since GATT Client sent a Exchange MTU Request PDU to the GATT Server.<br>**Event Data:**<br>st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP |
| BLE_GATTC_EVENT_CHAR_READ_RSP | When the read of Characteristic/Characteristic Descriptor has been completed, this event is notified to the application layer.<br>**Event Code: 0x400B**<br>**result:** |

<table>
<tr><td></td><td>
<table>
<tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr>
<tr><td>BLE_ERR_RSP_TIMEOUT(0x0011)</td><td>30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadCharUsingUuid() to the GATT Server.</td></tr>
</table>
Event Data:<br>st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_READ_RSP
</td></tr>
<tr><td>BLE_GATTC_EVENT_CHAR_PART_READ_RSP</td><td>
After calling R_BLE_GATTC_ReadLongChar(), this event notifies the application layer that the partial contents of Long Characteristic/Long Characteristic Descriptor has been received from the GATT Server.<br><br>
Event Code: 0x400D<br><br>
result:
<table>
<tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr>
<tr><td>BLE_ERR_RSP_TIMEOUT(0x0011)</td><td>30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadLongChar() to the GATT Server.</td></tr>
</table>
Event Data:<br>st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_PART_READ_RSP
</td></tr>
<tr><td>BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP</td><td>
This event notifies the application layer that the read of multiple Characteristics has been completed.<br><br>
Event Code: 0x400F
</td></tr>
</table>

| | |
|---|---|
| | **result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadMultiChar() to the GATT Server.<br><br>**Event Data:**<br><br>st_ble_gattc_rd_multi_char_evt_tBLE_GATTC_EVENT_MULTI_CHAR_READ_RSP |
| BLE_GATTC_EVENT_CHAR_WRITE_RSP | This event notifies the application layer that the write of Characteristic/Characteristic Descriptor has been completed.<br><br>**Event Code: 0x4013**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a write response since GATT Client sent a request for write by R_BLE_GATTC_WriteChar() to the GATT Server.<br><br>**Event Data:**<br><br>st_ble_gattc_wr_char_evt_tBLE_GATTC_EVENT_CHAR_WRITE_RSP |
| BLE_GATTC_EVENT_CHAR_WRITE_WITHOUT_RSP | This event notifies the application layer that the write without response of Characteristic/Characteristic Descriptor has been completed.<br><br>**Event Code: 0x4014** |

| | |
|---|---|
| | **result:**<br><br>BLE_SUCCESS(0x0000) — Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a write response since GATT Client sent a request for write by R_BLE_GTL_GATTC_WriteCharWithoutRsp() to the GATT Server.<br><br>**Event Data:**<br><br>st_ble_gattc_wr_char_evt_t |
| BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP | This event notifies the application layer of the one of the following.<br><br>• A segmentation to be written to Long Characteristic/Long Characteristic Descriptor has been sent to the GATT Server.<br>• The data written to one Characteristic by Reliable Writes has been sent to the GATT Server.<br><br>**Event Code: 0x4017**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) — Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a response since GATT Client sent a request for segmentation write by R_BLE_GATTC_WriteLongChar(), or 1 Characteristic write by R_BLE_GATTC_ReliableWrites() to the GATT Server. |

| | |
|---|---|
| | **Event Data:**<br><br>st_ble_gattc_char_part_wr_evt_tBLE_GATTC_EVENT_CHAR_PART_WRITE_RSP |
| BLE_GATTC_EVENT_HDL_VAL_NTF | This event notifies the application layer that a Notification has been received from a GATT Server.<br><br>**Event Code: 0x401B**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gattc_ntf_evt_tBLE_GATTC_EVENT_HDL_VAL_NTF |
| BLE_GATTC_EVENT_HDL_VAL_IND | This event notifies the application layer that a Indication has been received from a GATT Server.<br><br>When the GATT Client has received a Indication, host stack automatically sends a Confirmation to the GATT Server.<br><br>**Event Code: 0x401D**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) Insufficient resource is needed to generate the confirmation packet.<br><br>**Event Data:**<br><br>st_ble_gattc_ind_evt_tBLE_GATTC_EVENT_HDL_VAL_IND |
| BLE_GATTC_EVENT_HDL_VAL_SVC_CHNG | This event notifies the application layer that a service changed indication has been received from a GATT Server.<br><br>When the GATT Client has received a Indication from a Service Changed characteristic. |

| | |
|---|---|
| | **Event Code: 0x401E**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_MEM_ALLOC_FAILED(0x000C) Insufficient resource is needed to generate the confirmation packet.<br>**Event Data:**<br>st_ble_gattc_svc_chng_evt_tBLE_GATTC_EVENT_HDL_VAL_SVC_CHNG |
| BLE_GATTC_EVENT_CONN_IND | This event notifies the application layer that the link with the GATT Server has been established.<br>**Event Code: 0x4081**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_conn_evt_tBLE_GATTC_EVENT_CONN_IND |
| BLE_GATTC_EVENT_DISCONN_IND | This event notifies the application layer that the link with the GATT Server has been disconnected.<br>**Event Code: 0x4082**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_disconn_evt_tBLE_GATTC_EVENT_DISCONN_IND |
| BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND | This event notifies the application layer that 16-bit UUID Primary Service has been discovered. |

| | |
|---|---|
| | **Event Code: 0x40E0**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_serv_16_evt_tBLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND |
| BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND | This event notifies the application layer that 128-bit UUID Primary Service has been discovered.<br>**Event Code: 0x40E1**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_serv_128_evt_tBLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND |
| BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP | When the Primary Service discovery by R_BLE_GATTC_DiscAllPrimServ() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40E2**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP |
| BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP | When the Primary Service discovery by R_BLE_GATTC_DiscPrimServ() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40E3**<br>**result:** |

| | |
|---|---|
| | BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>noneBLE_GATTC_EVENT_PRIM_SERV_DISC_COMP |
| BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND | This event notifies the application layer that 16-bit UUID Secondary Service has been discovered.<br><br>**Event Code: 0x40E4**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gattc_serv_16_evt_tBLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND |
| BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND | This event notifies the application layer that 128-bit UUID Secondary Service has been discovered.<br><br>**Event Code: 0x40E5**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_gattc_serv_128_evt_tBLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND |
| BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP | When the Primary Service discovery by R_BLE_GATTC_DiscAllSecondServ() has been completed, this event is notified to the application layer.<br><br>**Event Code: 0x40E6**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:** |

| | |
|---|---|
| | noneBLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP |
| BLE_GATTC_EVENT_INC_SERV_16_DISC_IND | This event notifies the application layer that Included Service that includes 16-bit UUID Service has been discovered.<br>**Event Code: 0x40E7**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_inc_serv_16_evt_tBLE_GATTC_EVENT_INC_SERV_16_DISC_IND |
| BLE_GATTC_EVENT_INC_SERV_128_DISC_IND | This event notifies the application layer that Included Service that includes 128-bit UUID Service has been discovered.<br>**Event Code: 0x40E8**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_inc_serv_128_evt_tBLE_GATTC_EVENT_INC_SERV_128_DISC_IND |
| BLE_GATTC_EVENT_INC_SERV_DISC_COMP | When the Included Service discovery by R_BLE_GATTC_DiscIncServ() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40E9**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_INC_SERV_DISC_COMP |
| BLE_GATTC_EVENT_CHAR_16_DISC_IND | This event notifies the application layer that 16-bit UUID Characteristic has been |

| | |
|---|---|
| | discovered.<br>**Event Code: 0x40EA**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_char_16_evt_tBLE_GATTC_EVENT_CHAR_16_DISC_IND |
| BLE_GATTC_EVENT_CHAR_128_DISC_IND | This event notifies the application layer that 128-bit UUID Characteristic has been discovered.<br>**Event Code: 0x40EB**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_char_128_evt_tBLE_GATTC_EVENT_CHAR_128_DISC_IND |
| BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP | When the Characteristic discovery by R_BLE_GATTC_DiscAllChar() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40EC**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_ALL_CHAR_DISC_COMP |
| BLE_GATTC_EVENT_CHAR_DISC_COMP | When the Characteristic discovery by R_BLE_GATTC_DiscCharByUuid() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40ED** |

| | |
|---|---|
| | **result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_CHAR_DISC_COMP |
| BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND | This event notifies the application layer that 16-bit UUID Characteristic Descriptor has been discovered.<br>**Event Code: 0x40EE**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_char_desc_16_evt_tBLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND |
| BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND | This event notifies the application layer that 128-bit UUID Characteristic Descriptor has been discovered.<br>**Event Code: 0x40EF**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_char_desc_128_evt_tBLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND |
| BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP | When the Characteristic Descriptor discovery by R_BLE_GATTC_DiscAllCharDesc() has been completed, this event is notified to the application layer.<br>**Event Code: 0x40F0**<br>**result:**<br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| | **Event Data:**<br>noneBLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP |
| BLE_GATTC_EVENT_LONG_CHAR_READ_COMP | After calling R_BLE_GATTC_ReadLongChar(), this event notifies the application layer that all of the contents of the Characteristic/Long Characteristic Descriptor has been received from the GATT Server.<br>**Event Code: 0x40F1**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_LONG_CHAR_READ_COMP |
| BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP | This event notifies that the application layer that the write of Long Characteristic/Long Characteristic Descriptor has been completed.<br>**Event Code: 0x40F2**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a response since GATT Client sent a request for write by R_BLE_GATTC_WriteLongChar() to the GATT Server.<br>**Event Data:**<br>noneBLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP |
| BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP | This event notifies that the application layer that the GATT Server has received the data to be written to the Characteristics.<br>**Event Code: 0x40F3** |

| | |
|---|---|
| | **result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP |
| BLE_GATTC_EVENT_RELIABLE_WRITES_COMP | This event notifies the application layer that the Reliable Writes has been completed.<br>**Event Code: 0x40F4**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a response since GATT Client sent a request for execute write by R_BLE_GATTC_ReliableWrites() to the GATT Server.<br>**Event Data:**<br>st_ble_gattc_reliable_writes_comp_evt_tBLE_GATTC_EVENT_RELIABLE_WRITES_COMP |
| BLE_GATTC_EVENT_EXECUTE_WRITE_COMP | This event notifies the application layer that the Execute Write Command has been completed.<br>**Event Code: 0x40F5**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_reliable_writes_comp_evt_tBLE_GATTC_EVENT_RELIABLE_WRITES_COMP |
| BLE_GATTC_EVENT_ATT_INFO_REQ_IND | This event notifies the application layer that an Information Request Indication has been |

| | |
|---|---|
| | received.<br>**Event Code: 0x40F6**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_gattc_reliable_writes_comp_evt_tBLE_GATTC_EVENT_ATT_INFO_REQ_IND |
| BLE_GATTC_EVENT_INVALID | Invalid GATT Client Event.<br>**Event Code: 0x40FF**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>noneBLE_GATTC_EVENT_INVALID |

### Function Documentation

#### ◆ R_BLE_GATTC_Init()

ble_status_t R_BLE_GATTC_Init ( uint8_t *cb_num*)

This function initializes the GATT Client and registers the number of the callbacks for GATT Client event.

Specify the cb_num parameter to a value between 1 and BLE_GATTC_MAX_CB.
R_BLE_GATTC_RegisterCb() registers the callback.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb_num | The number of callbacks to be registered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The cb_num parameter is out of range. |

### ◆ R_BLE_GATTC_RegisterCb()

ble_status_t R_BLE_GATTC_RegisterCb ( ble_gattc_app_cb_t *cb*, uint8_t *priority* )

This function registers a callback function for GATT Client event.

The number of the callback that may be registered by this function is the value specified by R_BLE_GATTC_Init().
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb | Callback function for GATT Client event. |
| [in] | priority | The priority of the callback function.<br>Valid range is 1 <= priority <= BLE_GATTC_MAX_CB.<br>A lower priority number means a higher priority level. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The cb parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The priority parameter is out of range. |
| BLE_ERR_CONTEXT_FULL(0x000B) | Host stack has already registered the maximum number of callbacks. |

### ◆ R_BLE_GATTC_DeregisterCb()

ble_status_t R_BLE_GATTC_DeregisterCb ( ble_gattc_app_cb_t *cb*)

This function deregisters the callback function for GATT Client event.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb | The callback function to be deregistered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The cb parameter is specified as NULL. |
| BLE_ERR_NOT_FOUND(0x000D) | The callback has not been registered. |

### ◆ R_BLE_GATTC_ReqExMtu()

ble_status_t R_BLE_GATTC_ReqExMtu ( uint16_t *conn_hdl*, uint16_t *mtu* )

This function sends a MTU Exchange Request PDU to a GATT Server in order to change the current MTU.

MTU Exchange Response is notified by BLE_GATTC_EVENT_EX_MTU_RSP event.
The new MTU is the minimum value of the mtu parameter specified by this function and the mtu field in BLE_GATTC_EVENT_EX_MTU_RSP event. Default MTU size is 23 bytes.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be sent. |
| [in] | mtu | The maximum size(in bytes) of the GATT PDU that GATT Client can receive.<br>Valid range is 23 <= mtu <= 247. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The mtu parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscAllPrimServ()

ble_status_t R_BLE_GATTC_DiscAllPrimServ ( uint16_t *conn_hdl*)

This function discovers all Primary Services in a GATT Server.

When 16-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer.
When the Primary Service discovery has been completed,
BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_OPERATION(0x0009) | This function was called while processing other request. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscPrimServ()

ble_status_t R_BLE_GATTC_DiscPrimServ ( uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type* )

This function discovers Primary Service specified by p_uuid in a GATT Server.

When Primary Service whose uuid is the same as the specified uuid has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event or
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer.
When the Primary Service discovery has been completed,
BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |
| [in] | p_uuid | UUID of Primary Service to be discovered. |
| [in] | uuid_type | UUID type(16-bit or 128-bit).<br>macro: description<br>BLE_GATT_16_BIT_UUID_FORMAT(0x01): 16-bit UUID<br>BLE_GATT_128_BIT_UUID_FORMAT(0x02): 128-bit UUID |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_uuid parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The uuid_type parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscAllSecondServ()

ble_status_t R_BLE_GATTC_DiscAllSecondServ ( uint16_t *conn_hdl*)

This function discovers all Secondary Services in a GATT Server.

When a 16-bit UUID Secondary Service has been discovered,
BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND event is notified to the application layer.
When a 128-bit UUID Secondary Service has been discovered,
BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND event is notified to the application layer.
When the Secondary Service discovery has been completed,
BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscIncServ()

ble_status_t R_BLE_GATTC_DiscIncServ ( uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range* )

This function discovers Included Services within the specified attribute handle range in a GATT Server.

When Included Service that includes 16-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_16_DISC_IND event is notified to the application layer.
When Included Service that includes 128-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_128_DISC_IND event is notified to the application layer.
When the Included Service discovery has been completed, BLE_GATTC_EVENT_INC_SERV_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |
| [in] | p_range | Retrieval range of Included Service. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_range parameter is specified as NULL. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscAllChar()

ble_status_t R_BLE_GATTC_DiscAllChar ( uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range* )

This function discovers Characteristic within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.
When the Characteristic discovery has been completed, BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |
| [in] | p_range | Retrieval range of Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_range parameter is specified as NULL. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscCharByUuid()

ble_status_t R_BLE_GATTC_DiscCharByUuid ( uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type*, st_ble_gatt_hdl_range_t * *p_range* )

This function discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.
When the Characteristic discovery has been completed, BLE_GATTC_EVENT_CHAR_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |
| [in] | p_uuid | UUID of Characteristic to be discovered. |
| [in] | uuid_type | UUID type of Characteristic to be discovered.<br>macro: BLE_GATT_16_BIT_UUID_FORMAT(0x01) — The p_uuid parameter is 16-bit UUID.<br>macro: BLE_GATT_128_BIT_UUID_FORMAT(0x02) — The p_uuid parameter is 128-bit UUID. |
| [in] | p_range | Retrieval range of Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_uuid parameter or the p_range parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The uuid_type parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_DiscAllCharDesc()

ble_status_t R_BLE_GATTC_DiscAllCharDesc ( uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range* )

This function discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic Descriptor has been discovered, BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND event is notified to the application layer. When 128-bit UUID Characteristic Descriptor has been discovered, BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND event is notified to the application layer. When the Characteristic Descriptor discovery has been completed, BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be discovered. |
| [in] | p_range | Retrieval range of Characteristic Descriptor. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_range parameter is specified as NULL. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ReadChar()

ble_status_t R_BLE_GATTC_ReadChar ( uint16_t *conn_hdl*, uint16_t *value_hdl* )

This function reads a Characteristic/Characteristic Descriptor in a GATT Server.

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_RSP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be read. |
| [in] | value_hdl | Value handle of the Characteristic/Characteristic Descriptor to be read. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | 0 is specified in the value_hdl parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ReadCharUsingUuid()

ble_status_t R_BLE_GATTC_ReadCharUsingUuid ( uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type*, st_ble_gatt_hdl_range_t * *p_range* )

This function reads a Characteristic in a GATT Server using a specified UUID.

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle that identifies Characteristic to be read to GATT Server. |
| [in] | p_uuid | UUID of the Characteristic to be read. |
| [in] | uuid_type | UUID type of the Characteristic to be read.<br>macro: BLE_GATT_16_BIT_UUID_FORMAT(0x01) — The p_uuid parameter is 16-bit UUID.<br>macro: BLE_GATT_128_BIT_UUID_FORMAT(0x02) — The p_uuid parameter is 128-bit UUID. |
| [in] | p_range | Retrieval range of Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_uuid parameter or the p_range parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The uuid_type parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ReadLongChar()

ble_status_t R_BLE_GATTC_ReadLongChar ( uint16_t *conn_hdl*, uint16_t *value_hdl*, uint16_t *offset* )

This function reads a Long Characteristic in a GATT Server.

The contents of the Long Characteristic that has been read is notified every MTU-1 bytes to the application layer by BLE_GATTC_EVENT_CHAR_READ_RSP event.
When all of the contents has been received in GATT Client,
BLE_GATTC_EVENT_LONG_CHAR_READ_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be read. |
| [in] | value_hdl | Value handle of the Long Characteristic to be read. |
| [in] | offset | Offset that indicates the location to be read. Normally, set 0 to this parameter. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | 0 is specified in the value_hdl parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ReadMultiChar()

ble_status_t R_BLE_GATTC_ReadMultiChar ( uint16_t *conn_hdl*, st_ble_gattc_rd_multi_req_param_t * *p_list* )

This function reads multiple Characteristics in a GATT Server.

The contents of the multiple Characteristics that has been read is notified to the application layer by BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle that identifies Characteristic to be read to GATT Server. |
| [in] | p_list | List of Value Handles that point the Characteristics to be read. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_list parameter or the p_hdl_list field in the p_list parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | 0 is specified in the list_count field in the p_list parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_WriteCharWithoutRsp()

ble_status_t R_BLE_GATTC_WriteCharWithoutRsp ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data* )

This function writes a Characteristic in a GATT Server without response.

The result is returned from the API.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle that identifies Characteristic to be read to GATT Server. |
| [in] | p_write_data | Value to be written to the Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• 0 is specified in the value_len field in the p_value field in the p_write_data parameter.<br>• 0 is specified in the attr_hdl field in the p_write_data parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_SignedWriteChar()

ble_status_t R_BLE_GATTC_SignedWriteChar ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data* )

This function writes Signed Data to a Characteristic in a GATT Server without response.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be written. |
| [in] | p_write_data | Signed Data to be written to the Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• 0 is specified in the value_len field in the value field in the p_write_data parameter.<br>• 0 is specified in the attr_hdl field in the p_write_data parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function or Signed Data. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_WriteChar()

ble_status_t R_BLE_GATTC_WriteChar ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data* )

This function writes a Characteristic in a GATT Server.

The result of the write is notified in BLE_GATTC_EVENT_CHAR_WRITE_RSP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be written. |
| [in] | p_write_data | Value to be written to the Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• 0 is specified in the value_len field in the value field in the p_write_data parameter.<br>• 0 is specified in the attr_hdl field in the p_write_data parameter. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_WriteLongChar()

ble_status_t R_BLE_GATTC_WriteLongChar ( uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data*, uint16_t *offset* )

This function writes a Long Characteristic in a GATT Server.

The result of a write that has been done every segmentation is notified to the application layer in BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event.
The maximum writable size to a Long Characteristic with this function is 512 bytes.
When all of the contents has been written to the Long Characteristic,
BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be written. |
| [in] | p_write_data | Value to be written to the Long Characteristic. |
| [in] | offset | Offset that indicates the location to be written.<br>Normally, set 0 to this parameter.<br>If this parameter sets to a value other than 0, adjust the offset parameter and the length of the value to be written not to exceed the length of the Long Characteristic. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• The value_len field in the value field in the p_write_data parameter is 0.<br>• The sum of the value_len field in the value field in the p_write_data parameter and the offset parameter larger than 512.<br>• The attr_hdl field in the p_write_data parameter is 0. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ReliableWrites()

ble_status_t R_BLE_GATTC_ReliableWrites ( uint16_t *conn_hdl*, st_ble_gattc_reliable_writes_char_pair_t * *p_char_pair*, uint8_t *pair_num*, uint8_t *auto_flag* )

This function performs the Reliable Writes procedure described in GATT Specification.

When the data written to the Characteristic has been transmitted, BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event is notified to the application layer.
If the data included in the event is different from the data that GATT Client has sent, host stack automatically cancels the Reliable Writes.
After all of the contents has been sent to the GATT Server, if the auto_flag parameter has been set to BLE_GATTC_EXEC_AUTO, the GATT Server automatically writes the data to the Characteristic.
If the auto_flag parameter has been set to BLE_GATTC_EXEC_NOT_AUTO, BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer in GATT Client that all of the contents has been sent to the GATT Server. Then GATT Client requests for writing the data to the Characteristic to the GATT Server with R_BLE_GATTC_ExecWrite().
When the write has been done, BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the GATT Server to be written. |
| [in] | p_char_pair | Pair of Characteristic Value and Characteristic Value Handle identifying the Characteristic to be written by Reliable Writes. |
| [in] | pair_num | The number of the pairs specified by the p_char_pair parameter.<br>Valid range is 0 < pair_num <= BLE_GATTC_RELIABLE_WRITES_MAX_CHAR_PAIR. |
| [in] | auto_flag | The flag that indicates whether auto execution or not.<br>macro: BLE_GATTC_EXEC_AUTO(0x01) — description: Auto execution.<br>macro: BLE_GATTC_EXEC_NOT_AUTO (0x02) — description: Not auto execution. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The reason for this error is as follows:<br>• The p_char_pair parameter is specified as NULL.<br>• The p_value field in the value field in the write_data field in the p_char_pair parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The reason for this error is as follows:<br>• The pair_num parameter or the auto_flag parameter is out of range.<br>• The value_len field in the value field in the write_data field in the p_char_pair parameter is 0. |
| BLE_ERR_INVALID_OPERATION(0x0009) | While processing other request, this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function or to store the temporary write data. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

### ◆ R_BLE_GATTC_ExecWrite()

ble_status_t R_BLE_GATTC_ExecWrite ( uint16_t *conn_hdl*, uint8_t *exe_flag* )

If the auto execute of Reliable Writes is not specified by R_BLE_GATTC_ReliableWrites(), this function is used to execute a write to Characteristic.

When all of the contents has been sent to the GATT Server,
BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer.
After this event has been received, execute the write by this function.
The result of the write is notified by BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the target GATT Server. |
| [in] | exe_flag | The flag that indicates whether execution or cancellation. |

| macro | description |
|---|---|
| BLE_GATTC_EXECUTE_ | Execute the write. |

| | | |
|---|---|---|
| | | WRITE_CANCEL_FLAG(0x00)<br>BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG(0x01) Cancel the write. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The exe_flag parameter is out of range. |
| BLE_ERR_INVALID_OPERATION(0x0009) | The reason for this error is as follows:<br>• GATT Client has not requested for Reliable Writes by R_BLE_GATTC_ReliableWrites().<br>• Although auto execution has been specified by R_BLE_GATTC_ReliableWrites(), this function was called. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The GATT Server specified by conn_hdl was not found. |

## L2CAP

Interfaces » Networking » BLE Interface

### Functions

| | |
|---|---|
| ble_status_t | R_BLE_L2CAP_RegisterCfPsm (ble_l2cap_cf_app_cb_t cb, uint16_t psm, uint16_t lwm)<br>This function registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event. More... |
| ble_status_t | R_BLE_L2CAP_DeregisterCfPsm (uint16_t psm)<br>This function stops the use of the L2CAP CBFC Channel specified by |

| | |
|---|---|
| | the psm parameter and deregisters the callback function for L2CAP event. More... |
| ble_status_t | R_BLE_L2CAP_ReqCfConn (uint16_t conn_hdl, st_ble_l2cap_conn_req_param_t *p_conn_req_param)<br>This function sends a connection request for L2CAP CBFC Channel. More... |
| ble_status_t | R_BLE_L2CAP_RspCfConn (st_ble_l2cap_conn_rsp_param_t *p_conn_rsp_param)<br>This function replies to the connection request for L2CAP CBFC Channel from the remote device. More... |
| ble_status_t | R_BLE_L2CAP_DisconnectCf (uint16_t lcid)<br>This function sends a disconnection request for L2CAP CBFC Channel. More... |
| ble_status_t | R_BLE_L2CAP_SendCfCredit (uint16_t lcid, uint16_t credit)<br>This function sends credit to a remote device. More... |
| ble_status_t | R_BLE_L2CAP_SendCfData (uint16_t conn_hdl, uint16_t lcid, uint16_t data_len, uint8_t *p_sdu)<br>This function sends the data to a remote device via L2CAP CBFC Channel. More... |

## Detailed Description

### Data Structures

| | |
|---|---|
| struct | st_ble_l2cap_conn_req_param_t<br>L2CAP CBFC Channel connection request parameters. More... |
| struct | st_ble_l2cap_conn_rsp_param_t<br>L2CAP CBFC Channel connection response parameters. More... |
| struct | st_ble_l2cap_cf_conn_evt_t<br>L2CAP CBFC Channel connection parameters. More... |

| | |
|---|---|
| struct | st_ble_l2cap_cf_data_evt_t<br>Sent/Received Data parameters. More... |
| struct | st_ble_l2cap_cf_credit_evt_t<br>Credit parameters of local or remote device. More... |
| struct | st_ble_l2cap_cf_disconn_evt_t<br>Disconnection parameters. More... |
| struct | st_ble_l2cap_rej_evt_t<br>Command Reject parameters. More... |
| struct | st_ble_l2cap_cf_evt_data_t<br>st_ble_l2cap_cf_evt_data_t is the type of the data notified in a L2CAP Event. More... |

## Macros

| | |
|---|---|
| #define | BLE_L2CAP_MAX_CBFC_PSM<br>The maximum number of callbacks that host stack can register. |
| #define | BLE_L2CAP_CF_RSP_SUCCESS<br>Notify the remote device that the connection can be established. |
| #define | BLE_L2CAP_CF_RSP_RFSD_INSF_AUTH<br>Notify the remote device that the connection can not be established because of insufficient authentication. |
| #define | BLE_L2CAP_CF_RSP_RFSD_INSF_AUTRZ<br>Notify the remote device that the connection can not be established because of insufficient Authorization. |
| #define | BLE_L2CAP_CF_RSP_RFSD_INSF_ENC_KEY<br>Notify the remote device that the connection can not be established because of Encryption Key Size. |

| | |
|---|---|
| #define | BLE_L2CAP_CF_RSP_RFSD_INSF_ENC<br>Notify the remote device that the connection can not be established because of Encryption. |
| #define | BLE_L2CAP_CF_RSP_RFSD_UNAC_PARAM<br>Notify the remote device that the connection can not be established because the parameters is unacceptable to local device. |

### Typedefs

| | |
|---|---|
| typedef void(* | ble_l2cap_cf_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_l2cap_cf_evt_data_t *p_event_data)<br>ble_l2cap_cf_app_cb_t is the L2CAP Event callback function type. More... |

### Enumerations

| | |
|---|---|
| enum | e_r_ble_l2cap_cf_evt_t<br>L2CAP Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_l2cap_conn_req_param_t

struct st_ble_l2cap_conn_req_param_t

L2CAP CBFC Channel connection request parameters.

Data Fields

| | | |
|---|---|---|
| uint16_t | local_psm | Identifier indicating the protocol/profile that uses L2CAP CBFC Channel on local device. |
| uint16_t | remote_psm | Identifier indicating the protocol/profile that uses L2CAP CBFC Channel on remote device. |
| uint16_t | mtu | MTU size(byte) receivable on L2CAP CBFC Channel. |
| uint16_t | mps | MPS size(byte) receivable on L2CAP CBFC Channel. |
| uint16_t | credit | The number of LE-Frame that local device can receive. |

### ◆ st_ble_l2cap_conn_rsp_param_t

struct st_ble_l2cap_conn_rsp_param_t

L2CAP CBFC Channel connection response parameters.

Data Fields

| | | |
|---|---|---|
| uint16_t | lcid | CID identifying the L2CAP CBFC Channel on local device. The valid range is 0x40-0x40 + BLE_L2CAP_MAX_CBFC_PSM - 1. |
| uint16_t | response | The response to the connection request. Select one of the following. |

| macro | description |
|---|---|
| BLE_L2CAP_CF_RSP_SUCCESS(0x0000) | Notify the remote device that the connection can be established. |
| BLE_L2CAP_CF_RSP_RFSD_INSF_AUTH(0x0005) | Notify the remote device that the connection can not be established because of insufficient authentication. |
| BLE_L2CAP_CF_RSP_RFSD_INSF_AUTRZ(0x0006) | Notify the remote device that the connection can not be established because of insufficient Authorization. |
| BLE_L2CAP_CF_RSP_RFSD_INSF_ENC_KEY(0x0007) | Notify the remote device that the connection can not be established because of Encryption Key Size. |
| BLE_L2CAP_CF_RSP_RFSD_INSF_ENC(0x0008) | Notify the remote device that the connection can not be |

| | | |
|---|---|---|
| | | established because of Encryption.<br>BLE_L2CAP_CF_RSP_RFSD_UNAC_PARAM(0x000B) Notify the remote device that the connection can not be established because the parameters is unacceptable to local device. |
| uint16_t | mtu | MTU(byte) of packet that L2CAP CBFC Channel on local device can receive. |
| uint16_t | mps | MPS(byte) of packet that L2CAP CBFC Channel on local device can receive. |
| uint16_t | credit | The number of LE-Frame that L2CAP CBFC Channel on local device can receive. |

### ◆ st_ble_l2cap_cf_conn_evt_t

| struct st_ble_l2cap_cf_conn_evt_t | | |
|---|---|---|
| L2CAP CBFC Channel connection parameters. | | |
| Data Fields | | |
| uint16_t | cid | CID identifying the L2CAP CBFC Channel. |
| uint16_t | psm | PSM allocated by the cid field. |
| uint16_t | mtu | MTU of local (specified in BLE_L2CAP_EVENT_CF_CONN_IND) / remote (notified in BLE_L2CAP_EVENT_CF_CONN_CNF) device. |
| uint16_t | mps | MPS of local (specified in BLE_L2CAP_EVENT_CF_CONN_IND) / remote (notified in BLE_L2CAP_EVENT_CF_CONN_CNF) device. |
| uint16_t | credit | Credit of local (specified in BLE_L2CAP_EVENT_CF_CONN_IND) / remote (notified in BLE_L2CAP_EVENT_CF_CONN_CNF) device. |

### ◆ st_ble_l2cap_cf_data_evt_t

| struct st_ble_l2cap_cf_data_evt_t | | |
|---|---|---|
| Sent/Received Data parameters. | | |
| Data Fields | | |
| uint16_t | cid | CID identifying the L2CAP CBFC Channel that has sent or received the data . |
| uint16_t | psm | PSM allocated by the cid field. |
| uint16_t | data_len | Data length. |
| uint8_t * | p_data | Sent/Received data. |

### ◆ st_ble_l2cap_cf_credit_evt_t

| struct st_ble_l2cap_cf_credit_evt_t | | |
|---|---|---|
| Credit parameters of local or remote device. | | |
| Data Fields | | |
| uint16_t | cid | CID identifying the L2CAP CBFC Channel. |
| uint16_t | psm | PSM allocated by the cid field. |
| uint16_t | credit | Current credit of local/remote device. |

### ◆ st_ble_l2cap_cf_disconn_evt_t

| struct st_ble_l2cap_cf_disconn_evt_t | | |
|---|---|---|
| Disconnection parameters. | | |
| Data Fields | | |
| uint16_t | cid | CID identifying the L2CAP CBFC Channel that has been disconnected. |

### ◆ st_ble_l2cap_rej_evt_t

| struct st_ble_l2cap_rej_evt_t | | |
|---|---|---|
| Command Reject parameters. | | |
| Data Fields | | |
| uint16_t | reason | The reason that the remote device has sent Command Reject. |
| uint16_t | data_1 | Optional information about the reason that the remote device has sent Command Reject. |
| uint16_t | data_2 | Optional information about the |

| | | |
|---|---|---|
| | | reason that the remote device has sent Command Reject. |

### ◆ st_ble_l2cap_cf_evt_data_t

| struct st_ble_l2cap_cf_evt_data_t | | |
|---|---|---|
| st_ble_l2cap_cf_evt_data_t is the type of the data notified in a L2CAP Event. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle identifying the remote device. |
| uint16_t | param_len | The size of L2CAP Event parameters. |
| void * | p_param | L2CAP Event parameters. This parameter differs in each L2CAP Event. |

## Typedef Documentation

### ◆ ble_l2cap_cf_app_cb_t

ble_l2cap_cf_app_cb_t

ble_l2cap_cf_app_cb_t is the L2CAP Event callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | event_type | The type of L2CAP Event. |
| [in] | event_result | The result of L2CAP Event |
| [in] | p_event_data | Data notified by L2CAP Event. |

**Returns**

none

## Enumeration Type Documentation

### ◆ e_r_ble_l2cap_cf_evt_t

enum e_r_ble_l2cap_cf_evt_t

L2CAP Event Identifier.

| Enumerator | |
|---|---|
| BLE_L2CAP_EVENT_CF_CONN_CNF | After the connection request for L2CAP CBFC Channel has been sent with R_BLE_L2CAP_ReqCfConn(), when the L2CAP CBFC Channel connection response has been received, BLE_L2CAP_EVENT_CF_CONN_CNF event occurs.<br>**Event Code: 0x5001**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_RSP_TIMEOUT(0x0011) L2CAP Command timeout.<br>BLE_ERR_L2CAP_PSM_NOT_SUPPORTED(0x4002) PSM specified by R_BLE_L2CAP_ReqCfConn() is not supported.<br>BLE_ERR_L2CAP_NO_RESOURCE(0x4004) No resource for connection.<br>BLE_ERR_L2CAP_INSUF_AUTHEN(0x4005) Insufficient authentication.<br>BLE_ERR_L2CAP_INSUF_AUTHOR(0x4006) Insufficient authorization.<br>BLE_ERR_L2CAP_INSUF_ENC_KEY_SIZE(0x4007) Insufficient encryption key size.<br>BLE_ERR_L2CAP_REFUSE_INSUF_ENC(0x4008) Insufficient encryption.<br>BLE_ERR_L2CAP_REFUSE_INVALID_SCID(0x4009) Invalid Source CID.<br>BLE_ERR_L2CAP_REFUSE_SCID_ALREADY_ALLOC(0x400A) Source CID already allocated. |

| | |
|---|---|
| | BLE_ERR_L2CAP_REFUSE_UNACCEPTABLE_PARAM(0x400B) Unacceptable parameters.<br><br>**Event Data:**<br><br>st_ble_l2cap_cf_conn_evt_t |
| BLE_L2CAP_EVENT_CF_CONN_IND | When a connection request for L2CPA CBFC Channel has been received from a remote device, BLE_L2CAP_EVENT_CF_CONN_IND event occurs.<br><br>**Event Code: 0x5002**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>BLE_ERR_NOT_FOUND(0x000D) CF connection request has not been received or lcid not found.<br><br>BLE_ERR_L2CAP_PSM_NOT_SUPPORTED(0x4002) PSM specified by R_BLE_L2CAP_ReqCfConn() is not supported.<br><br>**Event Data:**<br><br>st_ble_l2cap_cf_conn_evt_t |
| BLE_L2CAP_EVENT_CF_DISCONN_CNF | After local device has sent a disconnection request for L2CAP CBFC Channel by R_BLE_L2CAP_DisconnectCf(), when the local device has received the response, BLE_L2CAP_EVENT_CF_DISCONN_CNF event occurs.<br><br>**Event Code: 0x5003**<br><br>**result:**<br><br>BLE_SUCCESS(0x0000) Success<br><br>**Event Data:**<br><br>st_ble_l2cap_cf_disconn_evt_t |
| BLE_L2CAP_EVENT_CF_DISCONN_IND | When local device has received a disconnection request for L2CAP CBFC Channel |

| | |
|---|---|
| | from the remote device, BLE_L2CAP_EVENT_CF_DISCONN_IND event occurs.<br>Host stack automatically replies the to the disconnection request.<br>**Event Code: 0x5004**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_l2cap_cf_disconn_evt_t |
| BLE_L2CAP_EVENT_CF_RX_DATA_IND | When local device has received data on L2CAP CBFC Channel, BLE_L2CAP_EVENT_CF_RX_DATA_IND event occurs.<br>**Event Code: 0x5005**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_l2cap_cf_data_evt_t |
| BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND | When the credit of the L2CAP CBFC Channel has reached the Low Water Mark, BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND event occurs.<br>**Event Code: 0x5006**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_l2cap_cf_credit_evt_t |
| BLE_L2CAP_EVENT_CF_TX_CRD_IND | When local device has received credit from a remote device, BLE_L2CAP_EVENT_CF_TX_CRD_IND event occurs. |

| | |
|---|---|
| | **Event Code: 0x5007**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_l2cap_cf_credit_evt_t |
| BLE_L2CAP_EVENT_CF_TX_DATA_CNF | When the data transmission has been completed from host stack to Controller, BLE_L2CAP_EVENT_CF_TX_DATA_CNF event occurs.<br>**Event Code: 0x5008**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_DISCONNECTED(0x000F) While transmitting data, L2CAP CBFC Channel has been disconnected.<br>**Event Data:**<br>st_ble_l2cap_cf_data_evt_t |
| BLE_L2CAP_EVENT_CMD_REJ | When local device has received Command Reject PDU, BLE_L2CAP_EVENT_CMD_REJ event occurs.<br>**Event Code: 0x5009**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_l2cap_rej_evt_t |

## Function Documentation

### ◆ R_BLE_L2CAP_RegisterCfPsm()

ble_status_t R_BLE_L2CAP_RegisterCfPsm ( ble_l2cap_cf_app_cb_t *cb*, uint16_t *psm*, uint16_t *lwm* )

This function registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event.

Only one callback is available per PSM. Configure in each PSM the Low Water Mark of the LE-Frames that the local device can receive.
When the number of the credit reaches the Low Water Mark, BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND event is notified to the application layer.
The number of PSM is defined as BLE_L2CAP_MAX_CBFC_PSM.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | cb | Callback function for L2CAP event. |
| [in] | psm | Identifier indicating the protocol/profile that uses L2CAP CBFC Channel. |

| type | range | description |
|---|---|---|
| Fixed, SIG assigned | 0x0001 - 0x007F | PSM defined by SIG. For more information on PSM, refer Bluetooth SIG Assigned Number ( https://www.bluetooth.com/specifications/assigned-numbers ). |
| Dynamic | 0x0080 - 0x00FF | Statically allocated PSM by custom protocol or dynamicall |

| | | |
|---|---|---|
| | | y allocated PSM by GATT Service. |
| [in] | lwm | Low Water Mark that indicates the LE-Frame numbers that the local device can receive. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The cb parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The psm parameter is out of range. |
| BLE_ERR_CONTEXT_FULL(0x000B) | More than BLE_L2CAP_MAX_CBFC_PSM+1 PSMs, callbacks has been registered. |

### ◆ R_BLE_L2CAP_DeregisterCfPsm()

ble_status_t R_BLE_L2CAP_DeregisterCfPsm ( uint16_t *psm*)

This function stops the use of the L2CAP CBFC Channel specified by the psm parameter and deregisters the callback function for L2CAP event.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | psm | PSM that is to be stopped to use the L2CAP CBFC Channel.<br>Set the PSM registered by R_BLE_L2CAP_RegisterCfPsm(). |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_NOT_FOUND(0x000D) | The callback function allocated by the psm parameter is not found. |

### ◆ R_BLE_L2CAP_ReqCfConn()

ble_status_t R_BLE_L2CAP_ReqCfConn ( uint16_t *conn_hdl*, st_ble_l2cap_conn_req_param_t * *p_conn_req_param* )

This function sends a connection request for L2CAP CBFC Channel.

The connection response is notified by BLE_L2CAP_EVENT_CF_CONN_CNF event.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device that the connection request is sent to. |
| [in] | p_conn_req_param | Connection request parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_conn_req_param parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The mtu parameter or the mps parameter is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | CF Channel connection has not been established. |
| BLE_ERR_CONTEXT_FULL(0x000B) | New CF Channel can not be registered or other L2CAP Command is processing. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | Insufficient memory is needed to generate this function. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by conn_hdl is not found. |
| BLE_ERR_NOT_YET_READY(0x0012) | The psm parameter is not registered. |

### ◆ R_BLE_L2CAP_RspCfConn()

ble_status_t R_BLE_L2CAP_RspCfConn ( st_ble_l2cap_conn_rsp_param_t * *p_conn_rsp_param*)

This function replies to the connection request for L2CAP CBFC Channel from the remote device.

The connection request is notified by BLE_L2CAP_EVENT_CF_CONN_IND event. The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_conn_rsp_param | Connection response parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_conn_rsp_param parameter is specified as NULL. |
| BLE_ERR_NOT_FOUND(0x000D) | A connection request for L2CAP CBFC Channel has not been received, or CID specified by the lcid field in the p_conn_rsp_param parameter is not found. |

### ◆ R_BLE_L2CAP_DisconnectCf()

| ble_status_t R_BLE_L2CAP_DisconnectCf ( uint16_t *lcid*) |
|---|

This function sends a disconnection request for L2CAP CBFC Channel.

When L2CAP CBFC Channel has been disconnected, BLE_L2CAP_EVENT_CF_DISCONN_CNF event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | lcid | CID identifying the L2CAP CBFC Channel that has been disconnected.<br>The valid range is 0x40 - (0x40 + BLE_L2CAP_MAX_CBFC_PSM - 1). |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_OPERATION(0x0009) | CF Channel connection has not been established. |
| BLE_ERR_CONTEXT_FULL(0x000B) | This function was called while processing other L2CAP command. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for L2CAP Command. |
| BLE_ERR_NOT_FOUND(0x000D) | CID specified the lcid parameter is not found. |

### ◆ R_BLE_L2CAP_SendCfCredit()

ble_status_t R_BLE_L2CAP_SendCfCredit ( uint16_t *lcid*, uint16_t *credit* )

This function sends credit to a remote device.

In L2CAP CBFC communication, if credit is 0, the remote device stops data transmission.
Therefore when processing the received data has been completed and local device affords to receive data, the remote device is notified of the number of LE-Frame that local device can receive by this function and local device can continue to receive data from the remote device.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | lcid | CID identifying the L2CAP CBFC Channel on local device that sends credit. |
| [in] | credit | Credit to be sent to the remote device. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The credit parameter is set to 0. |
| BLE_ERR_CONTEXT_FULL(0x000B) | This function was called while processing other L2CAP command. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for L2CAP Command. |

### ◆ R_BLE_L2CAP_SendCfData()

ble_status_t R_BLE_L2CAP_SendCfData ( uint16_t *conn_hdl*, uint16_t *lcid*, uint16_t *data_len*, uint8_t * *p_sdu* )

This function sends the data to a remote device via L2CAP CBFC Channel.

When the data transmission to Controller has been completed, BLE_L2CAP_EVENT_CF_TX_DATA_CNF event is notified to the application layer.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the remote device to be sent the data. |
| [in] | lcid | CID identifying the L2CAP CBFC Channel on local device used in the data transmission. |
| [in] | data_len | Length of the data. |
| [in] | p_sdu | Service Data Unit.<br>Input the data length specified by the data_len parameter to the first 2 bytes (Little Endian). |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_data parameter is specified as NULL. |
| BLE_ERR_INVALID_ARG(0x0003) | The length parameter is out of range. |
| BLE_ERR_INVALID_STATE(0x0008) | CF Channel connection has not been established or the data whose length exceeds the MTU has been sent. |
| BLE_ERR_ALREADY_IN_PROGRESS(0x000A) | Data transmission has been already started. |
| BLE_ERR_CONTEXT_FULL(0x000B) | L2CAP task queue is full. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for L2CAP Command. |
| BLE_ERR_NOT_FOUND(0x000D) | CID specified the lcid parameter is not found. |
| BLE_ERR_INVALID_HDL(0x000E) | The remote device specified by the conn_hdl parameter is not found. |

# VS

Interfaces » Networking » BLE Interface

## Functions

| | |
|---|---|
| ble_status_t | R_BLE_VS_Init (ble_vs_app_cb_t vs_cb)<br>This function initializes Vendor Specific API and registers a callback function for Vendor Specific Event. More... |
| ble_status_t | R_BLE_VS_StartTxTest (st_ble_vs_tx_test_param_t *p_tx_test_param)<br>This function starts extended Transmitter Test. More... |
| ble_status_t | R_BLE_VS_StartRxTest (st_ble_vs_rx_test_param_t *p_rx_test_param)<br>This function starts extended Receiver Test. More... |
| ble_status_t | R_BLE_VS_EndTest (void)<br>This function terminates the extended transmitter or receiver test. More... |
| ble_status_t | R_BLE_VS_SetTxPower (uint16_t conn_hdl, uint8_t tx_power)<br>This function configures transmit power. More... |
| ble_status_t | R_BLE_VS_GetTxPower (uint16_t conn_hdl)<br>This function gets transmit power. More... |
| ble_status_t | R_BLE_VS_SetCodingScheme (uint8_t coding_scheme)<br>This function configure default Coding scheme(S=8 or S=2) that is used in the case of selecting Coded PHY in Primary advertising PHY or Secondary advertising PHY advertising or request for link establishment. More... |
| ble_status_t | R_BLE_VS_SetRfControl (st_ble_vs_set_rf_ctrl_param_t *p_rf_ctrl)<br>This function performs power control on RF. More... |
| ble_status_t | R_BLE_VS_SetBdAddr (uint8_t area, st_ble_dev_addr_t *p_addr)<br>This function sets public/random address of local device to the area specified by the parameter. More... |

| | |
|---|---|
| ble_status_t | R_BLE_VS_GetBdAddr (uint8_t area, uint8_t addr_type)<br>This function gets currently configured public/random address. More... |
| ble_status_t | R_BLE_VS_GetRand (uint8_t rand_size)<br>This function generates 4-16 bytes of random number used in creating keys. More... |
| ble_status_t | R_BLE_VS_StartTxFlowEvtNtf (void)<br>This function starts the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow. More... |
| ble_status_t | R_BLE_VS_StopTxFlowEvtNtf (void)<br>This function stops the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow. More... |
| ble_status_t | R_BLE_VS_GetTxBufferNum (uint32_t *p_buffer_num)<br>This function retrieves the number of the available transmission packet buffers. More... |
| ble_status_t | R_BLE_VS_SetTxLimit (uint32_t tx_queue_lwm, uint32_t tx_queue_hwm)<br>This function sets the threshold for notifying the application layer of the TxFlow state. More... |
| ble_status_t | R_BLE_VS_SetScanChMap (uint16_t ch_map)<br>This function sets the scan channel map. More... |
| ble_status_t | R_BLE_VS_GetScanChMap (void)<br>This function gets currently scan channel map. More... |
| ble_status_t | R_BLE_VS_StartFirmwareUpdate (void)<br>This function starts the firmware update procedure. More... |
| ble_status_t | R_BLE_VS_SendFirmwareData (uint16_t index, uint16_t length, uint8_t const *const p_data) |

| | |
|---|---|
| | This function sends a firmware update data frame. More... |
| ble_status_t | R_BLE_VS_EndFirmwareUpdate (uint16_t end_index)<br>This function ends the firmware update procedure. More... |
| ble_status_t | R_BLE_VS_GetFirmwareVersion (void)<br>This function requests the BLE module firmware version. More... |
| ble_status_t | R_BLE_VS_RestartModule (void)<br>This function restarts the module. More... |

## Detailed Description

### Data Structures

| | |
|---|---|
| struct | st_ble_vs_tx_test_param_t<br>This is the extended transmitter test parameters used in R_BLE_VS_StartTxTest(). More... |
| struct | st_ble_vs_rx_test_param_t<br>This is the extended receiver test parameters used in R_BLE_VS_StartRxTest(). More... |
| struct | st_ble_vs_set_rf_ctrl_param_t<br>This is the RF parameters used in R_BLE_VS_SetRfControl(). More... |
| struct | st_ble_vs_test_end_evt_t<br>This structure notifies that the extended test has been terminated. More... |
| struct | st_ble_vs_set_tx_pwr_comp_evt_t<br>This structure notifies that tx power has been set. More... |
| struct | st_ble_vs_get_tx_pwr_comp_evt_t<br>This structure notifies that tx power has been retrieved. More... |

| | |
|---|---|
| struct | st_ble_vs_set_rf_ctrl_comp_evt_t<br>This structure notifies that RF has been configured. More... |
| struct | st_ble_vs_get_bd_addr_comp_evt_t<br>This structure notifies that BD_ADDR has been retrieved. More... |
| struct | st_ble_vs_get_rand_comp_evt_t<br>This structure notifies that random number has been generated. More... |
| struct | st_ble_vs_tx_flow_chg_evt_t<br>This structure notifies that the state transition of TxFlow has been changed. More... |
| struct | st_ble_vs_evt_data_t<br>st_ble_vs_evt_data_t is the type of the data notified in a Vendor Specific Event. More... |
| struct | st_ble_vs_get_scan_ch_map_comp_evt_t<br>This structure notifies that current scan channel map. More... |
| struct | st_ble_vs_get_fw_version_comp_evt_t<br>This structure notifies the current firmware version. More... |

## Macros

| | |
|---|---|
| #define | BLE_VS_TX_POWER_HIGH<br>High power level. |
| #define | BLE_VS_TX_POWER_MID<br>Middle power level. |
| #define | BLE_VS_TX_POWER_LOW<br>Low power level. |
| #define | BLE_VS_ADDR_AREA_REG |

| | |
|---|---|
| | Address in register is written or read. |
| #define | BLE_VS_ADDR_AREA_DATA_FLASH<br>Address in DataFlash is written or read. |
| #define | BLE_VS_EH_TX_PL_PRBS9<br>PRBS9 sequence '11111111100000111101..'. |
| #define | BLE_VS_EH_TX_PL_11110000<br>Repeated '11110000'. |
| #define | BLE_VS_EH_TX_PL_10101010<br>Repeated '10101010'. |
| #define | BLE_VS_EH_TX_PL_PRBS15<br>PRBS15 sequence. |
| #define | BLE_VS_EH_TX_PL_11111111<br>Repeated '11111111'. |
| #define | BLE_VS_EH_TX_PL_00000000<br>Repeated '00000000'. |
| #define | BLE_VS_EH_TX_PL_00001111<br>Repeated '00001111'. |
| #define | BLE_VS_EH_TX_PL_01010101<br>Repeated '01010101'. |
| #define | BLE_VS_EH_TEST_PHY_1M<br>1M PHY used in Transmitter/Receiver test. |
| #define | BLE_VS_EH_TEST_PHY_2M<br>2M PHY used in Transmitter/Receiver test. |

| | |
|---|---|
| #define | BLE_VS_EH_TEST_PHY_CODED<br>Coded PHY used in Receiver test. |
| #define | BLE_VS_EH_TEST_PHY_CODED_S_8<br>Coded PHY(S=8) used in Transmitter test. |
| #define | BLE_VS_EH_TEST_PHY_CODED_S_2<br>Coded PHY(S=2) used in Transmitter test. |
| #define | BLE_VS_RF_OFF<br>RF power off. |
| #define | BLE_VS_RF_ON<br>RF power on. |
| #define | BLE_VS_RF_INIT_PARAM_NOT_CHG<br>The parameters are not changed in RF power on. |
| #define | BLE_VS_RF_INIT_PARAM_CHG<br>The parameters are changed in RF power on. |
| #define | BLE_VS_CS_PRIM_ADV_S_8<br>Coding scheme for Primary Advertising PHY(S=8). |
| #define | BLE_VS_CS_PRIM_ADV_S_2<br>Coding scheme for Primary Advertising PHY(S=2). |
| #define | BLE_VS_CS_SECOND_ADV_S_8<br>Coding scheme for Secondary Advertising PHY(S=8). |
| #define | BLE_VS_CS_SECOND_ADV_S_2<br>Coding scheme for Secondary Advertising PHY(S=2). |

| | |
|---|---|
| #define | BLE_VS_CS_CONN_S_8<br>Coding scheme for request for link establishment(S=8). |
| #define | BLE_VS_CS_CONN_S_2<br>Coding scheme for request for link establishment(S=2). |
| #define | BLE_VS_TX_FLOW_CTL_ON<br>It means that the number of buffer has reached the High Water Mark from flow off state. |
| #define | BLE_VS_TX_FLOW_CTL_OFF<br>It means that the number of buffer has reached the Low Water Mark from flow on state. |
| #define | BLE_VS_SLEEP_OFF<br>Disable sleep mode in DA1453x. |
| #define | BLE_VS_EXTENDED_SLEEP<br>Set DA1453x in Extended Sleep mode. |
| #define | BLE_VS_DEEP_SLEEP<br>Set DA1453x in Deep Sleep mode. |
| #define | BLE_VS_HIBERNATION<br>Set DA1453x in Hibernation mode. |
| #define | BLE_VS_RAM_POWER_ON<br>Set DA1453x RAM is retained in Deep/Hibernation modes. |
| #define | BLE_VS_RAM_POWER_OFF<br>Set DA1453x is not retained in Deep/Hibernation modes. |

## Typedefs

| | |
|---|---|
| typedef void(* | ble_vs_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data) |

| | |
|---|---|
| | ble_vs_app_cb_t is the Vendor Specific Event callback function type. More... |

### Enumerations

| | |
|---|---|
| enum | e_r_ble_vs_evt_t<br>Vendor Specific Event Identifier. More... |

## Data Structure Documentation

### ◆ st_ble_vs_tx_test_param_t

| struct st_ble_vs_tx_test_param_t | | |
|---|---|---|
| This is the extended transmitter test parameters used in R_BLE_VS_StartTxTest(). | | |
| Data Fields | | |
| uint8_t | ch | Channel used in Tx test. |
| uint8_t | test_data_len | Length(in bytes) of the packet used in Tx Test. |
| uint8_t | packet_payload | Packet Payload. |
| uint8_t | phy | Transmitter PHY used in test. |
| uint8_t | tx_power | Tx Power Level used in DTM Tx Test. |
| uint8_t | option | Option. |
| uint16_t | num_of_packet | The number of packet to be sent. |

### ◆ st_ble_vs_rx_test_param_t

| struct st_ble_vs_rx_test_param_t | | |
|---|---|---|
| This is the extended receiver test parameters used in R_BLE_VS_StartRxTest(). | | |
| Data Fields | | |
| uint8_t | ch | Channel used in Rx test. |
| uint8_t | phy | Receiver PHY used in the test. |

### ◆ st_ble_vs_set_rf_ctrl_param_t

| struct st_ble_vs_set_rf_ctrl_param_t | | |
|---|---|---|
| This is the RF parameters used in R_BLE_VS_SetRfControl(). | | |
| Data Fields | | |
| uint8_t | power | RF power on/off. |
| uint8_t | option | This field indicates whether the |

| | | |
|---|---|---|
| | | parameters change in RF power on. |
| uint8_t | clval | RF rapid clock frequency adjust value(OSC internal CL adjust). |
| uint8_t | slow_clock | RF slow clock configurations. |
| uint8_t | tx_power | Set tx power in power on. |
| uint8_t | rf_option | Set RF option. |

### ◆ st_ble_vs_test_end_evt_t

| struct st_ble_vs_test_end_evt_t | | |
|---|---|---|
| This structure notifies that the extended test has been terminated. | | |
| Data Fields | | |
| uint16_t | num_of_packet | The number of packet successfully received in the receiver test. |
| uint16_t | num_of_crc_err_packet | The number of CRC error packets in the receiver test. |
| int8_t | ave_rssi | Average RSSI(dBm) in the receiver test. |
| int8_t | max_rssi | Maximum RSSI(dBm) in the receiver test. |
| int8_t | min_rssi | Minimum RSSI(dBm) in the receiver test. |

### ◆ st_ble_vs_set_tx_pwr_comp_evt_t

| struct st_ble_vs_set_tx_pwr_comp_evt_t | | |
|---|---|---|
| This structure notifies that tx power has been set. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle that identifying the link whose tx power has been set. |
| int8_t | curr_tx_pwr | Tx power that has been set(dBm). |

### ◆ st_ble_vs_get_tx_pwr_comp_evt_t

| struct st_ble_vs_get_tx_pwr_comp_evt_t | | |
|---|---|---|
| This structure notifies that tx power has been retrieved. | | |
| Data Fields | | |
| uint16_t | conn_hdl | Connection handle that identifying the link whose tx power has been retrieved. |

| | | |
|---|---|---|
| int8_t | curr_tx_pwr | Current tx power(dBm). |
| int8_t | max_tx_pwr | Maximum tx power(dBm). |

### ◆ st_ble_vs_set_rf_ctrl_comp_evt_t

| struct st_ble_vs_set_rf_ctrl_comp_evt_t | | |
|---|---|---|
| This structure notifies that RF has been configured. | | |
| Data Fields | | |
| uint8_t | ctrl | The result of RF power control. |

### ◆ st_ble_vs_get_bd_addr_comp_evt_t

| struct st_ble_vs_get_bd_addr_comp_evt_t | | |
|---|---|---|
| This structure notifies that BD_ADDR has been retrieved. | | |
| Data Fields | | |
| uint8_t | area | The area that public/random address has been retrieved.<br>value: description<br>BLE_VS_ADDR_AREA_REG(0x00): Register.<br>BLE_VS_ADDR_AREA_DATA_FLASH(0x01): Data Flash. |
| st_ble_dev_addr_t | addr | The address that has been retrieved. |

### ◆ st_ble_vs_get_rand_comp_evt_t

| struct st_ble_vs_get_rand_comp_evt_t | | |
|---|---|---|
| This structure notifies that random number has been generated. | | |
| Data Fields | | |
| uint8_t | rand_size | Length of random number. |
| uint8_t * | p_rand | Random number. |

### ◆ st_ble_vs_tx_flow_chg_evt_t

| struct st_ble_vs_tx_flow_chg_evt_t | | |
|---|---|---|
| This structure notifies that the state transition of TxFlow has been changed. | | |
| Data Fields | | |
| uint8_t | state | The state of the flow control.<br>value: description<br>BLE_VS_TX_FL: The number |

| | | |
|---|---|---|
| | | OW_CTL_ON(0x00) of buffer has reached the High Water Mark from flow off state.<br>BLE_VS_TX_FLOW_CTL_OFF(0x01) The number of buffer has reached the Low Water Mark from flow on state. |
| uint32_t | buffer_num | The number of the current transmission buffers. |

### ◆ st_ble_vs_evt_data_t

| struct st_ble_vs_evt_data_t | | |
|---|---|---|
| st_ble_vs_evt_data_t is the type of the data notified in a Vendor Specific Event. | | |
| Data Fields | | |
| uint16_t | param_len | The size of Vendor Specific Event parameters. |
| void * | p_param | Vendor Specific Event parameters. This parameter differs in each Vendor Specific Event. |

### ◆ st_ble_vs_get_scan_ch_map_comp_evt_t

| struct st_ble_vs_get_scan_ch_map_comp_evt_t | | |
|---|---|---|
| This structure notifies that current scan channel map. | | |
| Data Fields | | |
| uint8_t | ch_map | The result of current scan channel map. |

### ◆ st_ble_vs_get_fw_version_comp_evt_t

| struct st_ble_vs_get_fw_version_comp_evt_t | | |
|---|---|---|
| This structure notifies the current firmware version. | | |
| Data Fields | | |
| uint8_t | major | The result of get firmware version. |
| uint8_t | minor | |
| uint8_t | special | |

## Typedef Documentation

### ◆ ble_vs_app_cb_t

ble_vs_app_cb_t

ble_vs_app_cb_t is the Vendor Specific Event callback function type.

**Parameters**

| | | |
|---|---|---|
| [in] | event_type | The type of Vendor Specific Event. |
| [in] | event_result | The result of API call which generates the Vendor Specific Event. |
| [in] | p_event_data | Data notified in the Vendor Specific Event. |

**Returns**

none

## Enumeration Type Documentation

### ◆ e_r_ble_vs_evt_t

| enum e_r_ble_vs_evt_t | |
|---|---|
| Vendor Specific Event Identifier. | |
| Enumerator | |
| BLE_VS_EVENT_SET_TX_POWER | This event notifies that the tx power has been set by R_BLE_VS_SetTxPower().<br>**Event Code: 0x8001**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The tx_power parameter specified by R_BLE_VS_SetTxPower() is out of range.<br>BLE_ERR_INVALID_HDL(0x000E) The link identified with the conn_hdl specified by R_BLE_VS_SetTxPower() is not found.<br>**Event Data:**<br>st_ble_vs_set_tx_pwr_comp_evt_t |
| BLE_VS_EVENT_GET_TX_POWER | This event notifies that the tx power has been retrieved by R_BLE_VS_GetTxPower().<br>**Event Code: 0x8002**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_HDL(0x000E) The link identified with the conn_hdl specified by R_BLE_VS_GetTxPower() is not found.<br>**Event Data:**<br>st_ble_vs_get_tx_pwr_comp_evt_t |
| BLE_VS_EVENT_TX_TEST_START | This event notifies that the extended |

| | |
|---|---|
| | transmitter test has been started by R_BLE_VS_StartTxTest().<br>**Event Code: 0x8003**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The parameter specified by R_BLE_VS_StartTxTest() is out of range.<br>**Event Data:**<br>none |
| BLE_VS_EVENT_TX_TEST_TERM | This event notifies that the number specified by R_BLE_VS_StartTxTest() of packets has been sent.<br>**Event Code: 0x8004**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_RX_TEST_START | This event notifies that the extended receiver test has been started by R_BLE_VS_StartRxTest().<br>**Event Code: 0x8005**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The parameter specified by R_BLE_VS_StartRxTest() is out of range.<br>**Event Data:**<br>none |

<table>
<tr><td>BLE_VS_EVENT_TEST_END</td><td>This event notifies that the extended test has been terminated by R_BLE_VS_EndTest().<br><br>Event Code: 0x8006<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br><br>Event Data:<br><br>st_ble_vs_test_end_evt_t</td></tr>
<tr><td>BLE_VS_EVENT_SET_CODING_SCHEME_COMP</td><td>This event notifies that the coding scheme has been configured by R_BLE_VS_SetCodingScheme().<br><br>Event Code: 0x8007<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The coding_scheme parameter specified by R_BLE_VS_SetCodingScheme() is out of range.<br><br>Event Data:<br><br>none</td></tr>
<tr><td>BLE_VS_EVENT_RF_CONTROL_COMP</td><td>This event notifies that the RF has been configured by R_BLE_VS_SetRfControl().<br><br>Event Code: 0x8008<br><br>result:<br><br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The parameter specified by R_BLE_VS_SetRfControl() is out of range.<br>BLE_ERR_INVALID_OPERATION(0x000 During the power on or the power</td></tr>
</table>

| | | |
|---|---|---|
| | 9) | off, the same power state is specified by R_BLE_VS_SetRfControl(). |
| | **Event Data:** st_ble_vs_set_rf_ctrl_comp_evt_t | |
| BLE_VS_EVENT_SET_ADDR_COMP | This event notifies that public/random address has been set by R_BLE_VS_SetBdAddr(). **Event Code: 0x8009** **result:** | |
| | BLE_SUCCESS(0x0000) | Success |
| | BLE_ERR_INVALID_ARG(0x0003) | The area parameter or the type field in the p_addr parameter specified by R_BLE_VS_SetBdAddr() is out of range. |
| | **Event Data:** none | |
| BLE_VS_EVENT_GET_ADDR_COMP | This event notifies that public/random address has been retrieved by R_BLE_VS_GetBdAddr(). **Event Code: 0x800A** **result:** | |
| | BLE_SUCCESS(0x0000) | Success |
| | BLE_ERR_INVALID_ARG(0x0003) | The area parameter or the type field in the p_addr parameter specified by R_BLE_VS_GetBdAddr() is out of range. |
| | **Event Data:** st_ble_vs_get_bd_addr_comp_evt_t | |

| | |
|---|---|
| BLE_VS_EVENT_GET_RAND | This event notifies the application layer that random number has been generated by R_BLE_VS_GetRand().<br>**Event Code: 0x800B**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The rand_size parameter specified by R_BLE_VS_GetRand() is out of range.<br>**Event Data:**<br>st_ble_vs_get_rand_comp_evt_t |
| BLE_VS_EVENT_TX_FLOW_STATE_CHG | This event notifies the application layer of the state transition of TxFlow.<br>**Event Code: 0x800C**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_vs_tx_flow_chg_evt_t |
| BLE_VS_EVENT_FAIL_DETECT | This event notifies a failure occurs in RF. After receiving the event, reset MCU or RF.<br>**Event Code: 0x800D**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>None |
| BLE_VS_EVENT_SET_SCAN_CH_MAP | This event notifies that scan channel map has been set by R_BLE_VS_SetScanChMap().<br>**Event Code: 0x800E** |

| | |
|---|---|
| | **result:**<br>BLE_SUCCESS(0x0000) Success<br>BLE_ERR_INVALID_ARG(0x0003) The ch_map parameter specified by R_BLE_VS_SetScanChMap() is out of range.<br>**Event Data:**<br>none |
| BLE_VS_EVENT_GET_SCAN_CH_MAP | This event notifies that scan channel map has been retrieved by R_BLE_VS_GetScanChMap().<br>**Event Code: 0x800F**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_vs_get_scan_ch_map_comp_evt_t |
| BLE_VS_EVENT_START_FW_UPDATE_COMP | This event notifies that START_FW_UPDATE command has completed.<br>**Event Code: 0x072C**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_SEND_FW_DATA_COMP | This event notifies that SEND_FW_DATA command has completed.<br>**Event Code: 0x072D**<br>**result:**<br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| | **Event Data:**<br>none |
| BLE_VS_EVENT_END_FW_UPDATE_COMP | This event notifies that END_FW_UPDATE command has completed.<br>**Event Code: 0x072E**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_GET_FW_VERSION_COMP | This event notifies that END_FW_UPDATE command has completed.<br>**Event Code: 0x0772**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>st_ble_vs_get_fw_version_comp_evt_t |
| BLE_VS_EVENT_MODULE_READY_COMP | This event notifies that module is ready.<br>**Event Code: 0x0746**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_OTA_START_NOTIFY | This event notifies that OTA firmware update has been started.<br>**Event Code: 0x09B0**<br>**result:**<br>BLE_SUCCESS(0x0000) Success |

| | |
|---|---|
| | **Event Data:**<br>none |
| BLE_VS_EVENT_OTA_END_NOTIFY | This event notifies that OTA firmware update has been completed successfully.<br>**Event Code: 0x09C0**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_OTA_ERROR_NOTIFY | This event notifies that OTA firmware update has failed.<br>**Event Code: 0x09D0**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |
| BLE_VS_EVENT_INVALID | Invalid VS Event.<br>**Event Code: 0x80FF**<br>**result:**<br>BLE_SUCCESS(0x0000) Success<br>**Event Data:**<br>none |

### Function Documentation

### ◆ R_BLE_VS_Init()

ble_status_t R_BLE_VS_Init ( ble_vs_app_cb_t *vs_cb*)

This function initializes Vendor Specific API and registers a callback function for Vendor Specific Event.

The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [in] | vs_cb | Callback function to be registered. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The vs_cb parameter is specified as NULL. |
| BLE_ERR_CONTEXT_FULL(0x000B) | Callback function has already been registered. |

### ◆ R_BLE_VS_StartTxTest()

ble_status_t R_BLE_VS_StartTxTest ( st_ble_vs_tx_test_param_t * *p_tx_test_param*)

This function starts extended Transmitter Test.

The following extended transmitter test functions of DTM Tx are supported by this function.

- Tx Power
- Tx Modulation Enable/Modulation Disable
- Tx packet transmission/continuous transmission
- Tx packets count

The result of this API call is notified in BLE_VS_EVENT_TX_TEST_START event.
If the num_of_packet field in the p_tx_test_param parameter is other than 0x0000, BLE_VS_EVENT_TX_TEST_TERM event notifies the application layer that the number of packet has been sent.
If R_BLE_VS_EndTest() is called before the specified number of packets completions, BLE_VS_EVENT_TX_TEST_TERM event is not notified to the application layer.

The condition that phy field in the p_tx_test_param parameter is BLE_VS_EH_TEST_PHY_CODED_S_8(0x03) and option field is modulation(bit0:0) & continuous transmission(bit1:1) is not supported.

**Parameters**

| | | |
|---|---|---|
| [in] | p_tx_test_param | Tx Test parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_tx_test_param parameter is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_StartRxTest()

ble_status_t R_BLE_VS_StartRxTest ( st_ble_vs_rx_test_param_t * *p_rx_test_param*)

This function starts extended Receiver Test.

The result of this API call is notified in BLE_VS_EVENT_RX_TEST_START event. The following extended receiver test functions of DTM Rx are supported by this function.

- Calculating the maximum, the minimum and the average of RSSI in the receiver test.
- The number of CRC error packets in the receiver test.

The transmitter is configured to one of the following, the receiver can't receive the packets by this function.

- Tx Non-Modulation Enable
- Tx continuous transmission
  After R_BLE_VS_EndTest() has been called, the receiver test result value are notified in BLE_VS_EVENT_TEST_END event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_rx_test_param | The extended receiver test parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_rx_test_param parameter is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_EndTest()

ble_status_t R_BLE_VS_EndTest ( void )

This function terminates the extended transmitter or receiver test.

The result of this API call is notified in BLE_VS_EVENT_TEST_END event. In case of extended receiver test, this event notifies the application layer of the result of the extended receiver test.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_SetTxPower()

ble_status_t R_BLE_VS_SetTxPower ( uint16_t *conn_hdl*, uint8_t *tx_power* )

This function configures transmit power.

This function configures the following transmit power.

- The transmit power used in sending advertising PDU, scan request PDU, connection request PDU (in not connected state)
- The transmit power used in sending PDU in connected state. When configuring the transmit power used in not connected state, set the conn_hdl parameter to BLE_GAP_INIT_CONN_HDL(0xFFFF).
  When the transmit power used in connected state is configured, set the conn_hdl parameter to the connection handle of the link.
  Select one of the following transmit power levels.
- High
- Middle
- Low

Max transmit power of "High" is dependent on the configuration of the firmware. The result of this API call is notified in BLE_VS_EVENT_SET_TX_POWER event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose transmit power to be configured. If non connected state, set BLE_GAP_INIT_CONN_HDL(0xFFFF). |
| [in] | tx_power | Transmission power. Select one of the following.<br>• BLE_VS_TX_POWER_HIGH(0x00)<br>• BLE_VS_TX_POWER_MID(0x01)<br>• BLE_VS_TX_POWER_LOW(0x02) |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_GetTxPower()

ble_status_t R_BLE_VS_GetTxPower ( uint16_t *conn_hdl*)

This function gets transmit power.

This function gets the following transmit power.

- The transmit power used in sending advertising PDU, scan request PDU, connection request PDU (in not connected state)
- The transmit power used in sending PDU in connected state. When getting the transmit power used in not connected state, set the conn_hdl parameter to BLE_GAP_INIT_CONN_HDL(0xFFFF).
  When the transmit power used in connected state is retrieved, set the conn_hdl parameter to the connection handle of the link. The result of this API call is notified in BLE_VS_EVENT_GET_TX_POWER event.

**Parameters**

| | | |
|---|---|---|
| [in] | conn_hdl | Connection handle identifying the link whose transmit power to be retrieved. If non connected state, set BLE_GAP_INIT_CONN_HDL(0xFFFF). |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_SetCodingScheme()

ble_status_t R_BLE_VS_SetCodingScheme ( uint8_t *coding_scheme*)

This function configure default Coding scheme(S=8 or S=2) that is used in the case of selecting Coded PHY in Primary advertising PHY or Secondary advertising PHY advertising or request for link establishment.

After setting the default Coding scheme by this function, configure the advertising parameters by R_BLE_GAP_SetAdvParam() or send a request for link establishment.
The result of this API call is notified in BLE_VS_EVENT_SET_CODING_SCHEME_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | coding_scheme | Coding scheme for Primary advertising PHY, Secondary advertising PHY, request for link establishment.The coding_scheme field is set to a bitwise OR of the following values. |

| bit | description |
|---|---|
| bit0 | Coding scheme for Primary Advertising PHY(0:S=8/1:S=2). |
| bit1 | Coding scheme for Secondary Advertising PHY(0:S=8/1:S=2). |
| bit2 | Coding scheme for request for link establishment(0:S=8/1:S=2). |
| All other bits | Reserved for future use. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_SetRfControl()

ble_status_t R_BLE_VS_SetRfControl ( st_ble_vs_set_rf_ctrl_param_t * *p_rf_ctrl*)

This function performs power control on RF.

If BLE communication is not used for a long time, RF reduces the power consumption by moving to the RF Power-Down Mode.
When RF power on, RF initialization processing is executed.
After RF power off by this function, API functions other than this are not available until RF power on again.
The result of this API call is notified in BLE_VS_EVENT_RF_CONTROL_COMP event. After RF power on again with this function, call R_BLE_GAP_Terminate(), R_BLE_GAP_Init() in order to restart the host stack.

**Parameters**

| | | |
|---|---|---|
| [in] | p_rf_ctrl | RF parameters. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_rf_ctrl parameter is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_SetBdAddr()

ble_status_t R_BLE_VS_SetBdAddr ( uint8_t *area*, st_ble_dev_addr_t * *p_addr* )

This function sets public/random address of local device to the area specified by the parameter.

If the address is written in non-volatile area, the address is used as default address on the next MCU reset.
For more information on the random address, refer to Core Specification Vol 6, PartB, "1.3.2 Random Device Address".
The result of this API call is notified in BLE_VS_EVENT_SET_ADDR_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | area | The area that the address is to be written in.<br>Select one of the following.<br>macro: BLE_VS_ADDR_AREA_REG(0x00) — description: Address writing to non-volatile area is not performed. Only the address in register is written.<br>macro: BLE_VS_ADDR_AREA_DATA_FLASH(0x01) — description: Address wiring to DataFlash area is performed. |
| [in] | p_addr | The address to be set to the area. Set BLE_GAP_ADDR_PUBLIC(0x00) or BLE_GAP_ADDR_RAND(0x01) to the type field in the p_addr parameter. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_addr parameter is specified as NULL. |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_GetBdAddr()

ble_status_t R_BLE_VS_GetBdAddr ( uint8_t *area*, uint8_t *addr_type* )

This function gets currently configured public/random address.

The area parameter specifies the place where this function retrieves public/random address.
The result of this API call is notified in BLE_VS_EVENT_GET_ADDR_COMP event.

**Parameters**

| | | |
|---|---|---|
| [in] | area | The area that the address is to be retrieved.<br>Select one of the following.<br>macro / description<br>BLE_VS_ADDR_AREA_REG(0x00) — Retrieve the address in register.<br>BLE_VS_ADDR_AREA_DATA_FLASH(0x01) — Retrieve the address in DataFlash area. |
| [in] | addr_type | The address type that is type of the address to be retrieved.<br>macro / description<br>BLE_GAP_ADDR_PUBLIC(0x00) — Public address.<br>BLE_GAP_ADDR_RAND(0x01) — Random address. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_GetRand()

ble_status_t R_BLE_VS_GetRand ( uint8_t *rand_size*)

This function generates 4-16 bytes of random number used in creating keys.

The result of this API call is notified in BLE_VS_EVENT_GET_RAND event.

**Parameters**

| | | |
|---|---|---|
| [in] | rand_size | Length of the random number (byte).<br>The valid range is 4<=rand_size<=16. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_STATE(0x0008) | The task for host stack is not running. |
| BLE_ERR_MEM_ALLOC_FAILED(0x000C) | There are no memories for Vendor Specific Command. |

### ◆ R_BLE_VS_StartTxFlowEvtNtf()

ble_status_t R_BLE_VS_StartTxFlowEvtNtf ( void )

This function starts the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow.

If the number of the available transmission packet buffers is the following, BLE_VS_EVENT_TX_FLOW_STATE_CHG event notifies the application layer of the state of the TxFlow.

- The number of the available transmission packet buffers is less than Low Water Mark.
- The number of the available transmission packet buffers is more than High Water Mark. The result of this API call is returned by a return value.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_VS_StopTxFlowEvtNtf()

ble_status_t R_BLE_VS_StopTxFlowEvtNtf ( void )

This function stops the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow.

The result of this API call is returned by a return value.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |

### ◆ R_BLE_VS_GetTxBufferNum()

ble_status_t R_BLE_VS_GetTxBufferNum ( uint32_t * *p_buffer_num*)

This function retrieves the number of the available transmission packet buffers.

The maximum number of the available buffers is 10.
The result of this API call is returned by a return value.

**Parameters**

| | | |
|---|---|---|
| [out] | p_buffer_num | The number of the available transmission packet buffers. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_PTR(0x0001) | The p_buffer_num parameter is specified as NULL. |

### ◆ R_BLE_VS_SetTxLimit()

ble_status_t R_BLE_VS_SetTxLimit ( uint32_t *tx_queue_lwm*, uint32_t *tx_queue_hwm* )

This function sets the threshold for notifying the application layer of the TxFlow state.

Call this function before the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) has been started by R_BLE_VS_StartTxFlowEvtNtf().
The result is returned from this API.
Vendor Specific API supports the flow control function(TxFlow) for the transmission on L2CAP fixed channel in Basic Mode such as GATT.
Host stack has 10 transmission packet buffers for the transmission.
When the number of the available transmission packet buffers has been less than Low Water Mark, the state of TxFlow transmits into the TxFlow OFF state from the TxFlow ON state that is the initial state and host stack notifies the application layer of timing to stop packet transmission.
When host stack has sent the transmission packets to Controller and the number of the available transmission packet buffers has been more than High Water Mark, the state of TxFlow transmits into the TxFlow ON state from the TxFlow OFF state and host stack notifies the application layer of

timing to restart packet transmission.
It is possible to perform flow control on a fixed channel by using the event notification.

**Parameters**

| | | |
|---|---|---|
| [in] | tx_queue_lwm | Low Water Mark. Set 0-9 less than tx_queue_hwm to the parameter. When the number of the available transmission packet buffers has been less than the value specified by the tx_queue_lwm parameter, host stack notifies the application layer of the timing to stop packet transmission. |
| [in] | tx_queue_hwm | High Water Mark. Set 1-10 more than tx_queue_lwm to the parameter. When the number of the available transmission packet buffers has been more than the value specified by the tx_queue_hwm parameter, host stack notifies the application layer of the timing to restart packet transmission. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The tx_queue_lwm parameter or the tx_queue_hwm parameter is out of range. |

### ◆ R_BLE_VS_SetScanChMap()

ble_status_t R_BLE_VS_SetScanChMap ( uint16_t *ch_map*)

This function sets the scan channel map.

Set specify the scan channel for use.
At least one channel must be enabled.

***Note***

*Calling this API while Scan is already running will not change the channel map.*

**Parameters**

| | | |
|---|---|---|
| [in] | ch_map | Specify the channel map for use. |

| bit | description |
|---|---|
| bit0 | Enable channel 37 for use (0:disable, 1:enable) |
| bit1 | Enable channel 38 for use (0:disable, 1:enable) |
| bit2 | Enable channel 39 for use (0:disable, 1:enable) |
| All other bits | Reserved for future use. |

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG(0x0003) | The ch_map parameter is out of range. |

### ◆ R_BLE_VS_GetScanChMap()

| ble_status_t R_BLE_VS_GetScanChMap ( void ) |
|---|
| This function gets currently scan channel map.<br>The result of this API call is notified in BLE_VS_EVENT_GET_SCAN_CH_MAP event.<br><br>**Return values**<br>BLE_SUCCESS(0x0000) : Success |

### ◆ R_BLE_VS_StartFirmwareUpdate()

| ble_status_t R_BLE_VS_StartFirmwareUpdate ( void ) |
|---|
| This function starts the firmware update procedure.<br>The result of this API call is notified in BLE_VS_EVENT_START_FW_UPDATE_COMP event.<br><br>**Return values**<br>BLE_SUCCESS(0x0000) : Success<br>BLE_ERR_INVALID_MODE : A command was sent from an invalid mode<br>BLE_ERR_UNSUPPORTED : This API does not support |

### ◆ R_BLE_VS_SendFirmwareData()

ble_status_t R_BLE_VS_SendFirmwareData ( uint16_t *index*, uint16_t *length*, uint8_t const *const *p_data* )

This function sends a firmware update data frame.

**Parameters**

| | | |
|---|---|---|
| [in] | index | The index of the current data frame. |
| [in] | length | The length of the current data frame. |
| [in] | p_data | A pointer to the data frame. |

The result of this API call is notified in BLE_VS_EVENT_SEND_FW_DATA_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_ARG | An input parameter was invalid |
| BLE_ERR_INVALID_MODE | A command was sent from an invalid mode |
| BLE_ERR_UNSUPPORTED | This API does not support |

### ◆ R_BLE_VS_EndFirmwareUpdate()

ble_status_t R_BLE_VS_EndFirmwareUpdate ( uint16_t *end_index*)

This function ends the firmware update procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | end_index | The index of the last data frame. |

The result of this API call is notified in BLE_VS_EVENT_END_FW_UPDATE_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_MODE | A command was sent from an invalid mode |
| BLE_ERR_UNSUPPORTED | This API does not support |

### ◆ R_BLE_VS_GetFirmwareVersion()

| ble_status_t R_BLE_VS_GetFirmwareVersion ( void ) |
|---|

This function requests the BLE module firmware version.

The result of this API call is notified in BLE_VS_EVENT_GET_FW_VERSION_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_INVALID_MODE | A command was sent from an invalid mode |
| BLE_ERR_UNSUPPORTED | This API does not support |

### ◆ R_BLE_VS_RestartModule()

| ble_status_t R_BLE_VS_RestartModule ( void ) |
|---|

This function restarts the module.

The result of this API call is notified in BLE_VS_EVENT_MODULE_READY_COMP event.

**Return values**

| | |
|---|---|
| BLE_SUCCESS(0x0000) | Success |
| BLE_ERR_RSP_TIMEOUT | A command did not receive a response |
| BLE_ERR_INVALID_MODE | A command was sent from an invalid mode |
| BLE_ERR_UNSUPPORTED | This API does not support |

## 5.3.11.3 BLE Mesh Network Interfaces

Interfaces » Networking

### Detailed Description

BLE Mesh Network Interfaces.

### Modules

| | |
|---|---|
| | BLE Mesh Access Interface<br>Interface for BLE Mesh Access functions. |
| | BLE Mesh Bearer Interface |

| | |
|---|---|
| | Interface for BLE Mesh Bearer functions. |
| | BLE Mesh Bearer Platform Interface<br>Interface for BLE Mesh Bearer Platform functions. |
| | BLE Mesh Health Server Interface<br>Interface for BLE Mesh Model Health Server functions. |
| | BLE Mesh Interface<br>Interface for BLE Mesh functions. |
| | BLE Mesh Lower Trans Interface<br>Interface for BLE Mesh Lower Trans functions. |
| | BLE Mesh Model Client Interface<br>Interface for BLE Mesh Model Client functions. |
| | BLE Mesh Model Configuration Client Interface<br>Interface for BLE Mesh Model Configuration Client functions. |
| | BLE Mesh Model Server Interface<br>Interface for BLE Mesh Model Server functions. |
| | BLE Mesh Network Interface<br>Interface for BLE Mesh Network functions. |
| | BLE Mesh Provision Interface<br>Interface for BLE Mesh Provision functions. |
| | BLE Mesh Scene Server Interface<br>Interface for BLE Mesh Model Scene Server functions. |
| | BLE Mesh Upper Trans Interface<br>Interface for BLE Mesh Upper Trans functions. |

## BLE Mesh Access Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Access functions.

# Summary

The BLE Mesh Access interface for the BLE Mesh Network Access (BLE MESH ACCESS) peripheral provides BLE Mesh Network Access functionality.

### Modules

| | |
|---|---|
| | Application Callback |

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_access_model_id_t |
| union | rm_ble_mesh_access_model_id_t.__unnamed__ |
| struct | rm_ble_mesh_access_element_descriptor_t |
| struct | rm_ble_mesh_access_address_t |
| struct | rm_ble_mesh_access_publish_info_t |
| struct | rm_ble_mesh_access_model_req_msg_context_t |
| struct | rm_ble_mesh_access_req_msg_raw_t |
| struct | rm_ble_mesh_access_model_req_msg_t |
| struct | rm_ble_mesh_access_publish_setting_t |
| struct | rm_ble_mesh_access_pdu_t |
| struct | rm_ble_mesh_access_model_state_parameter_t |
| struct | rm_ble_mesh_access_extended_parameter_t |
| struct | rm_ble_mesh_access_device_entry_t |
| struct | rm_ble_mesh_access_model_t |

| | |
|---|---|
| struct | rm_ble_mesh_access_server_state_t |
| struct | rm_ble_mesh_access_provisioned_device_entry_t |
| struct | rm_ble_mesh_access_associated_keys_t |
| struct | rm_ble_mesh_access_friend_security_credential_info_t |
| struct | rm_ble_mesh_access_cfg_t |
| struct | rm_ble_mesh_access_api_t |
| struct | rm_ble_mesh_access_instance_t |

## Macros

| | |
|---|---|
| #define | RM_BLE_MESH_ACCESS_VADDR_LABEL_UUID_SIZE |
| #define | RM_BLE_MESH_ACCESS_NETKEY_SIZE |
| #define | RM_BLE_MESH_ACCESS_APPKEY_SIZE |
| #define | RM_BLE_MESH_ACCESS_KEY_SIZE |

## Typedefs

| | |
|---|---|
| typedef uint8_t | rm_ble_mesh_access_node_id_t |
| typedef uint8_t | rm_ble_mesh_access_element_handle_t |
| typedef uint16_t | rm_ble_mesh_access_model_handle_t |
| typedef uint16_t | rm_ble_mesh_access_model_id_sig_t |
| typedef uint32_t | rm_ble_mesh_access_model_id_vendor_t |
| typedef uint32_t | rm_ble_mesh_access_address_handle_t |
| typedef uint32_t | rm_ble_mesh_access_device_key_handle_t |
| typedef void | rm_ble_mesh_access_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_access_model_req_msg_type_t |
| enum | rm_ble_mesh_access_iv_update_test_mode_t |
| enum | rm_ble_mesh_access_message_opcode_t |

| | |
|---|---|
| enum | rm_ble_mesh_access_model_type_info_t |
| enum | rm_ble_mesh_access_model_id_info_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_access_model_id_t

| struct rm_ble_mesh_access_model_id_t | | |
|---|---|---|
| Model ID datatype | | |
| Data Fields | | |
| union rm_ble_mesh_access_model_id_t | __unnamed__ | Vendor/SIG ID |
| rm_ble_mesh_access_model_type_info_t | type | Model type - SIG or Vendor |

### ◆ rm_ble_mesh_access_model_id_t.__unnamed__

| union rm_ble_mesh_access_model_id_t.__unnamed__ |
|---|
| Vendor/SIG ID |

### ◆ rm_ble_mesh_access_element_descriptor_t

| struct rm_ble_mesh_access_element_descriptor_t | | |
|---|---|---|
| Element description format. | | |
| Data Fields | | |
| uint16_t | loc | Location descriptor |

### ◆ rm_ble_mesh_access_address_t

| struct rm_ble_mesh_access_address_t | | |
|---|---|---|
| Unicast/Virtual/Group Address. | | |
| Data Fields | | |
| uint8_t | use_label | Flag - which field to be used |
| rm_ble_mesh_network_address_t | addr | Address |
| uint8_t | label[RM_BLE_MESH_ACCESS_LABEL_UUID_LENGTH] | Label UUID |

### ◆ rm_ble_mesh_access_publish_info_t

| struct rm_ble_mesh_access_publish_info_t |
|---|
| Access Publication related information |
| Data Fields |

| | | |
|---|---|---|
| rm_ble_mesh_access_address_t | addr | PublishAddress (Unicast/Virtual/Group) |
| uint16_t | appkey_index | • AppKey Index (when set from remote).<br>• AppKey Handle (when set from locally for Configuration Client). |
| uint8_t | crden_flag | CredentialFlag |
| uint8_t | ttl | PublishTTL |
| uint8_t | period | PublishPeriod |
| uint8_t | rtx_count | PublishRetransmitCount |
| uint8_t | rtx_interval_steps | PublishRetransmitIntervalSteps |
| uint8_t | remote | Flag - if called from local or remote |

### ◆ rm_ble_mesh_access_model_req_msg_context_t

| struct rm_ble_mesh_access_model_req_msg_context_t | | |
|---|---|---|
| Context of message received for a specific model instance. This is required to send response appropriately. | | |
| Data Fields | | |
| rm_ble_mesh_access_model_handle_t | handle | Model Handle - for which request is received |
| rm_ble_mesh_network_address_t | saddr | Source Address - originator of request |
| rm_ble_mesh_network_address_t | daddr | Destination Address - of the request |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Associated Subnet Identifier |
| rm_ble_mesh_network_appkey_handle_t | appkey_handle | Associated AppKey Identifier |

### ◆ rm_ble_mesh_access_req_msg_raw_t

| struct rm_ble_mesh_access_req_msg_raw_t | | |
|---|---|---|
| Uninterpreted/raw received message for a specific model instance. | | |
| Data Fields | | |
| uint32_t | opcode | Request Opcode |
| uint8_t * | data_param | Raw received message |
| uint16_t | data_len | Raw received message length |

### ◆ rm_ble_mesh_access_model_req_msg_t

| |
|---|

| struct rm_ble_mesh_access_model_req_msg_t | | |
|---|---|---|
| Requested message type for a specific model instance. | | |
| Data Fields | | |
| rm_ble_mesh_access_model_req_msg_type_t | type | Flag: GET, SET or Others |
| uint8_t | to_be_acked | Flag: True or False |

### ◆ rm_ble_mesh_access_publish_setting_t

| struct rm_ble_mesh_access_publish_setting_t | | |
|---|---|---|
| Publish setting | | |
| Data Fields | | |
| uint8_t | ttl | Time to Live |
| uint8_t | to_publish | Flag to indicate if the message also to be published |

### ◆ rm_ble_mesh_access_pdu_t

| struct rm_ble_mesh_access_pdu_t | | |
|---|---|---|
| PDU setting | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | src_addr | 16 bit Source Address |
| rm_ble_mesh_network_address_t | dst_addr | 16 bit Destination Address |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Subnet Handle |
| rm_ble_mesh_network_appkey_handle_t | appkey_handle | AppKey Handle |
| uint8_t | ttl | Time to Live |
| uint32_t | opcode | Request Opcode |
| uint8_t * | data_param | Raw received message |
| uint16_t | data_len | Raw received message length |

### ◆ rm_ble_mesh_access_model_state_parameter_t

| struct rm_ble_mesh_access_model_state_parameter_t | | |
|---|---|---|
| Model specific state parameters in a request or response message | | |
| Data Fields | | |
| uint8_t | state_type | State Type |
| void * | state | State pointer |

### ◆ rm_ble_mesh_access_extended_parameter_t

| struct rm_ble_mesh_access_extended_parameter_t | | |
|---|---|---|
| Additional parameters in a Model specific request or response message | | |
| Data Fields | | |
| uint8_t | ext_type | State/Extended Type |
| void * | ext | State/Extended data structure pointer |

### ◆ rm_ble_mesh_access_device_entry_t

| struct rm_ble_mesh_access_device_entry_t | | |
|---|---|---|
| Provisioned Device List Data Structure, containing Primary Element Address and number of elements. | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | uaddr | Unicast address of the first element |
| uint8_t | num_elements | Number of Elements |

### ◆ rm_ble_mesh_access_model_t

| struct rm_ble_mesh_access_model_t | | |
|---|---|---|
| Data structure for model.<br>Models could be bluetooth SIG defined or vendor defined. | | |
| Data Fields | | |
| rm_ble_mesh_access_model_id_t | model_id | Model ID |
| rm_ble_mesh_access_model_callback_t | model_callback | Callback function pointer to receive packets from the underlying protocol layers |
| rm_ble_mesh_access_timeout_callback_t | timeout_callback | Callback function called when Publication Timer expires. Set to NULL if model does not support periodic publication. |
| uint16_t | num_opcodes | Number of Opcodes |
| const uint32_t * | opcodes | List of Opcodes |

### ◆ rm_ble_mesh_access_server_state_t

| struct rm_ble_mesh_access_server_state_t | | |
|---|---|---|
| API to send reply or to update state change | | |
| Data Fields | | |
| rm_ble_mesh_access_model_req_msg_context_t * | p_context | Context of the message. |

| | | |
|---|---|---|
| rm_ble_mesh_access_model_state_parameter_t * | p_current_state_parameter | Model specific current state parameters. |
| rm_ble_mesh_access_model_state_parameter_t * | p_target_state_parameter | Model specific target state parameters (NULL: to be ignored). |
| uint16_t | remaining_time | Time from current state to target state (0: to be ignored). |
| rm_ble_mesh_access_extended_parameter_t * | p_extended_parameter | Additional parameters (NULL: to be ignored). |
| uint8_t | reply | If unicast response to be sent. |
| uint8_t | publish | If state to be published. |

### ◆ rm_ble_mesh_access_provisioned_device_entry_t

| struct rm_ble_mesh_access_provisioned_device_entry_t | | |
|---|---|---|
| Provisioned Device List Data Structure, containing Primary Element Address and number of elements. | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | uaddr | Unicast address of the first element |
| uint8_t | num_elements | Number of Elements |

### ◆ rm_ble_mesh_access_associated_keys_t

| struct rm_ble_mesh_access_associated_keys_t | | |
|---|---|---|
| Associated key | | |
| Data Fields | | |
| uint8_t | privacy_key[RM_BLE_MESH_ACCESS_KEY_SIZE] | Privacy key |
| uint8_t | encrypt_key[RM_BLE_MESH_ACCESS_KEY_SIZE] | Encryption key |
| uint8_t | beacon_key[RM_BLE_MESH_ACCESS_KEY_SIZE] | Beacon key |
| uint8_t | is_new_key | Whether new key or not |

### ◆ rm_ble_mesh_access_friend_security_credential_info_t

| struct rm_ble_mesh_access_friend_security_credential_info_t | | |
|---|---|---|
| To add Security Credential of a LPN or the Friend. | | |
| Data Fields | | |
| rm_ble_mesh_network_address_ | lpn_addr | Address of the LPN. |

| | | |
|---|---|---|
| t | | |
| rm_ble_mesh_network_address_t | friend_addr | Address of the Friend. |
| uint16_t | lpn_counter | Number of Friend Request messages the LPN has sent. |
| uint16_t | friend_counter | Number of Friend Offer messages the Friend has sent. |

### ◆ rm_ble_mesh_access_cfg_t

| struct rm_ble_mesh_access_cfg_t | | |
|---|---|---|
| BLE MESH ACCESS configuration parameters. | | |
| Data Fields | | |
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware.<br><br>the parameters for initialization. |
| rm_ble_mesh_access_element_descriptor_t * | p_element_descriptor | Element description format. |
| rm_ble_mesh_upper_trans_instance_t const * | p_mesh_upper_trans_instance | Instance structure of upper trans. |
| rm_ble_mesh_access_element_handle_t | element_number | Element number. |
| void * | p_context | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| void const * | p_extend | Placeholder for user extension. |

### ◆ rm_ble_mesh_access_api_t

| struct rm_ble_mesh_access_api_t | |
|---|---|
| BLE MESH ACCESS functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_access_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | registerModel )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_t const *const p_model, rm_ble_mesh_access_model_handle_t *const p_model_handle) |

| | |
|---|---|
| | |
| fsp_err_t(* | getElementHandle )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t elem_addr, rm_ble_mesh_access_element_handle_t *const p_handle) |
| | |
| fsp_err_t(* | getElementHandleForModelHandle )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_element_handle_t *const p_elem_handle) |
| | |
| fsp_err_t(* | getModelHandle )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_element_handle_t elem_handle, rm_ble_mesh_access_model_id_t model_id, rm_ble_mesh_access_model_handle_t *const p_handle) |
| | |
| fsp_err_t(* | publish )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint8_t reliable) |
| | |
| fsp_err_t(* | reliablePublish )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint32_t rsp_opcode) |
| | |
| fsp_err_t(* | reply )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, uint8_t ttl, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw) |
| | |
| fsp_err_t(* | replyAndPublish )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw, rm_ble_mesh_access_publish_setting_t const *const p_publish_setting) |
| | |
| fsp_err_t(* | sendPdu )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_pdu_t const *const p_pdu, uint8_t reliable) |
| | |
| fsp_err_t(* | getCompositionData )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_buffer_t *const p_buffer) |
| | |
| fsp_err_t(* | reset )(rm_ble_mesh_access_ctrl_t *const p_ctrl) |

| | |
|---|---|
| | |
| fsp_err_t(* | getElementCount )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_count) |
| | |
| fsp_err_t(* | setPrimaryUnicastAddress )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| | |
| fsp_err_t(* | getPrimaryUnicastAddress )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t *const p_addr) |
| | |
| fsp_err_t(* | setDefaultTtl )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ttl) |
| | |
| fsp_err_t(* | getDefaultTtl )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_ttl) |
| | |
| fsp_err_t(* | setIvIndex )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t iv_index, uint8_t iv_update_flag) |
| | |
| fsp_err_t(* | getIvIndex )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t *const p_iv_index, uint8_t *const p_iv_update_flag) |
| | |
| fsp_err_t(* | getIvIndexByIvi )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ivi, uint32_t *const p_iv_index) |
| | |
| fsp_err_t(* | setFeaturesField )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t enable, uint8_t feature) |
| | |
| fsp_err_t(* | getFeaturesField )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_enable, uint8_t feature) |
| | |
| fsp_err_t(* | getFeatures )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_features) |
| | |
| fsp_err_t(* | getFriendshipRole )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_friend_role) |
| | |
| fsp_err_t(* | setFriendshipRole )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t friend_role) |

| | |
|---|---|
| | |
| fsp_err_t(* | addDeviceKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_dev_key, rm_ble_mesh_network_address_t uaddr, uint8_t num_elements) |
| | |
| fsp_err_t(* | getDeviceKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t dev_key_index, uint8_t **const p_dev_key) |
| | |
| fsp_err_t(* | removeAllDeviceKeys )(rm_ble_mesh_access_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | getProvisionedDeviceList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_provisioned_device_entry_t const *const p_prov_dev_list, uint16_t *const p_num_entries) |
| | |
| fsp_err_t(* | getDeviceKeyHandle )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t prim_elem_uaddr, rm_ble_mesh_access_device_key_handle_t *const p_handle) |
| | |
| fsp_err_t(* | getAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_appkey_handle_t appkey_handle, uint8_t **const p_app_key, uint8_t *const p_aid) |
| | |
| fsp_err_t(* | addUpdateNetkey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, uint32_t opcode, uint8_t const *const p_net_key) |
| | |
| fsp_err_t(* | addFriendSecurityCredential )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index, rm_ble_mesh_access_friend_security_credential_info_t info) |
| | |
| fsp_err_t(* | deleteFriendSecurityCredential )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index) |
| | |
| fsp_err_t(* | findSubnet )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle) |
| | |
| fsp_err_t(* | findMasterSubnet )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t friend_subnet_handle, |

| | |
|---|---|
| | rm_ble_mesh_network_subnet_handle_t *const p_master_subnet_handle) |
| | |
| fsp_err_t(* | deleteNetKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| | |
| fsp_err_t(* | getNetKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_net_key) |
| | |
| fsp_err_t(* | getNetKeyIndexList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_netkey_count, uint16_t *const p_netkey_index_list) |
| | |
| fsp_err_t(* | lookUpNid )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t nid, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set) |
| | |
| fsp_err_t(* | lookUpNetworkId )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_network_id, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set) |
| | |
| fsp_err_t(* | lookUpAid )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t aid, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle, uint8_t *const p_app_key) |
| | |
| fsp_err_t(* | setProvisioningData )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_provision_data_t const *const p_prov_data) |
| | |
| fsp_err_t(* | getSubnetNid )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_nid) |
| | |
| fsp_err_t(* | getSubnetPrivacyKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_privacy_key) |
| | |
| fsp_err_t(* | getSubnetNetworkId )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_network_id) |
| | |
| fsp_err_t(* | getSubnetBeaconKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_beacon_key) |
| | |
| fsp_err_t(* | getSubnetIdentityKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_identity_key) |
| | |
| fsp_err_t(* | getSubnetEncryptionKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_encrypt_key) |
| | |
| fsp_err_t(* | getNodeIdentity )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state) |
| | |
| fsp_err_t(* | setNodeIdentity )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state) |
| | |
| fsp_err_t(* | getKeyRefreshPhase )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_key_refresh_state) |
| | |
| fsp_err_t(* | setKeyRefreshPhase )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t const *const p_key_refresh_state) |
| | |
| fsp_err_t(* | setTransmitState )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t tx_state) |
| | |
| fsp_err_t(* | getTransmitState )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t *const p_tx_state) |
| | |
| fsp_err_t(* | addAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| | |
| fsp_err_t(* | updateAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | deleteAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key) |
| | |
| fsp_err_t(* | getAppKeyHandle )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle) |
| | |
| fsp_err_t(* | getAppKeyIndexList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list) |
| | |
| fsp_err_t(* | bindModelWithAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index) |
| | |
| fsp_err_t(* | unbindModelWithAppKey )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index) |
| | |
| fsp_err_t(* | getModelAppKeyList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list) |
| | |
| fsp_err_t(* | setModelPublication )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info) |
| | |
| fsp_err_t(* | setModelPublicationPeriodDivisor )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint8_t period_divisor) |
| | |
| fsp_err_t(* | getModelPublication )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info) |
| | |
| fsp_err_t(* | addModelSubscription )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr) |
| | |
| fsp_err_t(* | deleteModelSubscription )(rm_ble_mesh_access_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr) |
| | |
| fsp_err_t(* | deleteAllModelSubscription )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle) |
| | |
| fsp_err_t(* | getModelSubscriptionList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list) |
| | |
| fsp_err_t(* | getAllModelSubscriptionList )(rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list) |
| | |
| fsp_err_t(* | isValidElementAddress )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| | |
| fsp_err_t(* | isFixedGroupAddressToBeProcessed )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| | |
| fsp_err_t(* | isValidSubscriptionAddress )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr) |
| | |
| fsp_err_t(* | enableIvUpdateTestMode )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_iv_update_test_mode_t mode) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_access_api_t::open) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_cfg_t const *const p_cfg)

Open access middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_access_api_t::close) (rm_ble_mesh_access_ctrl_t *const p_ctrl)

Close access middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ registerModel

fsp_err_t(* rm_ble_mesh_access_api_t::registerModel) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_t const *const p_model, rm_ble_mesh_access_model_handle_t *const p_model_handle)

Register a model with the access layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_model | Pointer to model structure. |
| [out] | p_model_handle | Pointer to model handle. |

### ◆ getElementHandle

fsp_err_t(* rm_ble_mesh_access_api_t::getElementHandle) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t elem_addr, rm_ble_mesh_access_element_handle_t *const p_handle)

Get element handle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | elem_addr | Address of the corresponding element. |
| [out] | p_handle | Pointer to model handle. |

### ◆ getElementHandleForModelHandle

fsp_err_t(* rm_ble_mesh_access_api_t::getElementHandleForModelHandle) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_element_handle_t *const p_elem_handle)

Get element handle for a given model handle

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Model handle. |
| [out] | p_elem_handle | Pointer to element handle. |

### ◆ getModelHandle

fsp_err_t(* rm_ble_mesh_access_api_t::getModelHandle) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_element_handle_t elem_handle, rm_ble_mesh_access_model_id_t model_id, rm_ble_mesh_access_model_handle_t *const p_handle)

Get model handle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | elem_handle | Element identifier associated with the Model. |
| [in] | model_id | Model identifier for which the model handle to be searched. |
| [out] | p_handle | Pointer to model handle. |

### ◆ publish

fsp_err_t(* rm_ble_mesh_access_api_t::publish) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint8_t reliable)

API to publish access layer message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_handl | Pointer to model handle. |
| [in] | p_publish_message | Pointer to received message structure. |
| [in] | reliable | MS_TRUE for reliable message. MS_FALSE otherwise. |

### ◆ reliablePublish

fsp_err_t(* rm_ble_mesh_access_api_t::reliablePublish) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t const *const p_handle, rm_ble_mesh_access_req_msg_raw_t const *const p_publish_message, uint32_t rsp_opcode)

API to reliably publish access layer message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_handl | Pointer to model handle. |
| [in] | p_publish_message | Pointer to raw received message structure. |
| [in] | rsp_opcode | Response opcode. |

### ◆ reply

fsp_err_t(* rm_ble_mesh_access_api_t::reply) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, uint8_t ttl, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw)

API to reply to access layer message.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_req_msg_context | Pointer to context of received message structure. |
| [in] | ttl | Time to live. |
| [in] | p_req_msg_raw | Pointer to received message structure. |

### ◆ replyAndPublish

fsp_err_t(* rm_ble_mesh_access_api_t::replyAndPublish) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_req_msg_context_t const *const p_req_msg_context, rm_ble_mesh_access_req_msg_raw_t const *const p_req_msg_raw, rm_ble_mesh_access_publish_setting_t const *const p_publish_setting)

API to reply to access layer message and optionally also to publish.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_req_msg_context | Pointer to context of received message structure. |
| [in] | p_req_msg_raw | Pointer to received message structure. |
| [in] | p_publish_setting | Pointer to publish setting structure. |

### ◆ sendPdu

fsp_err_t(* rm_ble_mesh_access_api_t::sendPdu) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_pdu_t const *const p_pdu, uint8_t reliable)

API to send access PDUs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_pdu | Pointer to PDU structure. |
| [in] | reliable | If requires lower transport ACK, set reliable as TRUE. |

### ◆ getCompositionData

fsp_err_t(* rm_ble_mesh_access_api_t::getCompositionData) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_buffer_t *const p_buffer)

Get composition data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_buffer | Pointer to buffer structure. |

### ◆ reset

fsp_err_t(* rm_ble_mesh_access_api_t::reset) (rm_ble_mesh_access_ctrl_t *const p_ctrl)

To reset a node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ getElementCount

fsp_err_t(* rm_ble_mesh_access_api_t::getElementCount) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_count)

To get the number of elements in local node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_count | Pointer to number of elements. |

### ◆ setPrimaryUnicastAddress

fsp_err_t(* rm_ble_mesh_access_api_t::setPrimaryUnicastAddress) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr)

To set primary unicast address.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | Primary Unicast address to be set. |

### ◆ getPrimaryUnicastAddress

fsp_err_t(* rm_ble_mesh_access_api_t::getPrimaryUnicastAddress) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t *const p_addr)

To get primary unicast address.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_addr | Pointer to address. |

### ◆ setDefaultTtl

fsp_err_t(* rm_ble_mesh_access_api_t::setDefaultTtl) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ttl)

To set default TTL.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | ttl | Default TTL to be set. |

### ◆ getDefaultTtl

fsp_err_t(* rm_ble_mesh_access_api_t::getDefaultTtl) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_ttl)

To get default TTL.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_ttl | Pointer to TTL. |

### ◆ setIvIndex

fsp_err_t(* rm_ble_mesh_access_api_t::setIvIndex) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t iv_index, uint8_t iv_update_flag)

To set IV Index.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | iv_index | IV index to be set. |
| [in] | iv_update_flag | IV update flag. |

### ◆ getIvIndex

fsp_err_t(* rm_ble_mesh_access_api_t::getIvIndex) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint32_t *const p_iv_index, uint8_t *const p_iv_update_flag)

To get IV Index.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_iv_index | Pointer to index. |
| [out] | p_iv_update_flag | Pointer to update flag. |

### ◆ getIvIndexByIvi

fsp_err_t(* rm_ble_mesh_access_api_t::getIvIndexByIvi) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t ivi, uint32_t *const p_iv_index)

To get IV Index by IVI.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | ivi | Least significant bit of the IV index used in the once to authenticate and encrypt the network PDU. |
| [out] | p_iv_index | Pointer to index. |

### ◆ setFeaturesField

fsp_err_t(* rm_ble_mesh_access_api_t::setFeaturesField) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t enable, uint8_t feature)

To enable/disable a feature.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | enable | Enable or Disable. |
| [in] | feature | Relay, proxy, friend or low power. |

### ◆ getFeaturesField

fsp_err_t(* rm_ble_mesh_access_api_t::getFeaturesField) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_enable, uint8_t feature)

To get state of a feature.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_enable | Pointer to enable. |
| [in] | feature | Relay, proxy, friend or low power. |

### ◆ getFeatures

fsp_err_t(* rm_ble_mesh_access_api_t::getFeatures) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_features)

To get state of all features.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_features | Pointer to features. |

### ◆ getFriendshipRole

fsp_err_t(* rm_ble_mesh_access_api_t::getFriendshipRole) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t *const p_friend_role)

To get friendship role of the node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_friend_role | Pointer to friend role. |

### ◆ setFriendshipRole

fsp_err_t(* rm_ble_mesh_access_api_t::setFriendshipRole) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t friend_role)

To set friendship role of the node.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | friend_role | Friend role. |

### ◆ addDeviceKey

fsp_err_t(* rm_ble_mesh_access_api_t::addDeviceKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_dev_key, rm_ble_mesh_network_address_t uaddr, uint8_t num_elements)

To add Device Key.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_dev_key | Pointer to device Key. |
| [in] | uaddr | Unicast address of the first element. |
| [in] | num_elements | Number of elements. |

### ◆ getDeviceKey

fsp_err_t(* rm_ble_mesh_access_api_t::getDeviceKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t dev_key_index, uint8_t **const p_dev_key)

To get Device Key.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | dev_key_index | Device key index. |
| [out] | p_dev_key | Pointer to device key to be returned. |

### ◆ removeAllDeviceKeys

fsp_err_t(* rm_ble_mesh_access_api_t::removeAllDeviceKeys) (rm_ble_mesh_access_ctrl_t *const p_ctrl)

To remove all Device Keys.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ getProvisionedDeviceList

fsp_err_t(* rm_ble_mesh_access_api_t::getProvisionedDeviceList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_provisioned_device_entry_t const *const p_prov_dev_list, uint16_t *const p_num_entries)

To get list of Provisioned Device List.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_prov_dev_list | Pointer to provisioned device list structure. |
| [in] | p_num_entries | Pointer to number of entries. |

### ◆ getDeviceKeyHandle

fsp_err_t(* rm_ble_mesh_access_api_t::getDeviceKeyHandle) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t prim_elem_uaddr, rm_ble_mesh_access_device_key_handle_t *const p_handle)

To get Device Key Handle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | prim_elem_uaddr | Primary element address to be searched. |
| [in] | p_handle | Pointer to handle. |

### ◆ getAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::getAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_appkey_handle_t appkey_handle, uint8_t **const p_app_key, uint8_t *const p_aid)

To get AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | appkey_handle | AppKey Handle. |
| [out] | p_app_key | Pointer to AppKey to be returned. |
| [out] | p_aid | Pointer to AID to be returned. |

### ◆ addUpdateNetkey

fsp_err_t(* rm_ble_mesh_access_api_t::addUpdateNetkey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, uint32_t opcode, uint8_t const *const p_net_key)

To add/update NetKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | netkey_index | Identifies global index of NetKey. A 12bits value. |
| [in] | opcode | To identify Add or Update NetKey. |
| [out] | p_net_key | Pointer to NetKey. |

### ◆ addFriendSecurityCredential

fsp_err_t(* rm_ble_mesh_access_api_t::addFriendSecurityCredential) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index, rm_ble_mesh_access_friend_security_credential_info_t info)

To add Security Credential of a LPN or the Friend.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Identifies associated subnet. |
| [in] | friend_index | Friend Index. |
| [in] | info | Security credential of a LPN or the friend. |

### ◆ deleteFriendSecurityCredential

fsp_err_t(* rm_ble_mesh_access_api_t::deleteFriendSecurityCredential) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t friend_index)

To delete the Security Credential of a LPN or the Friend.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Identifies associated subnet. |
| [in] | friend_index | Friend index. |

### ◆ findSubnet

fsp_err_t(* rm_ble_mesh_access_api_t::findSubnet) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t netkey_index, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle)

To find a Subnet associated with the NetKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | netkey_index | Identifies global Index of NetKey, corresponding Subnet to be returned. |
| [out] | p_subnet_handle | Pointer to subnet handle. |

### ◆ findMasterSubnet

fsp_err_t(* rm_ble_mesh_access_api_t::findMasterSubnet) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t friend_subnet_handle, rm_ble_mesh_network_subnet_handle_t *const p_master_subnet_handle)

To find the Master Subnet associated with the friend security credential, identified by Friend Subnet Handle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | friend_subnet_handle | Identifies the friend subnet handle, corresponding to friend subnet handle. |
| [out] | p_master_subnet_handle | Pointer to master subnet handle. |

### ◆ deleteNetKey

fsp_err_t(* rm_ble_mesh_access_api_t::deleteNetKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle)

To delete NetKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the subnet for which NetKey to be deleted. |

### ◆ getNetKey

fsp_err_t(* rm_ble_mesh_access_api_t::getNetKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_net_key)

To get NetKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Target subnet handle to get net key. |
| [in] | p_net_key | Pointer to NetKey. |

### ◆ getNetKeyIndexList

fsp_err_t(* rm_ble_mesh_access_api_t::getNetKeyIndexList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_netkey_count, uint16_t *const p_netkey_index_list)

To get list of all known NetKeys.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_netkey_count | Pointer to NetKey count. |
| [in] | p_netkey_index_list | Pointer to NetKey index list. |

### ◆ lookUpNid

fsp_err_t(* rm_ble_mesh_access_api_t::lookUpNid) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t nid, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set)

To search for NID.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | nid | NID to be searched in all known subnets for match. |
| [out] | p_subnet_handle | Pointer to subnet handle. |
| [out] | p_key_set | Pointer to key set. |

### ◆ lookUpNetworkId

fsp_err_t(* rm_ble_mesh_access_api_t::lookUpNetworkId) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t const *const p_network_id, rm_ble_mesh_network_subnet_handle_t *const p_subnet_handle, rm_ble_mesh_access_associated_keys_t *const p_key_set)

To search for Network ID.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_network_id | Pointer to network ID to be searched in all known subnets for match. |
| [out] | p_subnet_handle | Pointer to subnet handle. |
| [out] | p_key_set | Pointer to key settings. |

### ◆ lookUpAid

fsp_err_t(* rm_ble_mesh_access_api_t::lookUpAid) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t aid, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle, uint8_t *const p_app_key)

To search for AID.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | aid | AID to be searched in all known AppKeys for match. |
| [out] | p_appkey_handle | Pointer to AppKey handle. |
| [out] | p_app_key | Pointer to AppKey. |

### ◆ setProvisioningData

fsp_err_t(* rm_ble_mesh_access_api_t::setProvisioningData) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_provision_data_t const *const p_prov_data)

Set Provisioning Data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_prov_data | Pointer to provisioning data structure. |

### ◆ getSubnetNid

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetNid) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_nid)

To get NID associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_nid | Pointer to NID. |

### ◆ getSubnetPrivacyKey

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetPrivacyKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_privacy_key)

To get privacy Key associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_privacy_key | Pointer to Privacy Key. |

### ◆ getSubnetNetworkId

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetNetworkId) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_network_id)

To get Network ID associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_network_id | Pointer to Network ID. |

### ◆ getSubnetBeaconKey

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetBeaconKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_beacon_key)

To get Beacon Key associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_beacon_key | Pointer to Beacon Key. |

### ◆ getSubnetIdentityKey

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetIdentityKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_identity_key)

To get Identity Key associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_identity_key | Pointer to Identity Key. |

### ◆ getSubnetEncryptionKey

fsp_err_t(* rm_ble_mesh_access_api_t::getSubnetEncryptionKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t handle, uint8_t *const p_encrypt_key)

To get Encryption Key associated with a subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Handle identifying the subnet. |
| [out] | p_encrypt_key | Pointer to Encryption Key. |

### ◆ getNodeIdentity

fsp_err_t(* rm_ble_mesh_access_api_t::getNodeIdentity) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state)

To get Node Identity.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle identifying the subnet. |
| [out] | p_id_state | Pointer to node identity state. |

### ◆ setNodeIdentity

fsp_err_t(* rm_ble_mesh_access_api_t::setNodeIdentity) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_id_state)

To set Node Identity.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle identifying the subnet. |
| [out] | p_id_state | Pointer to node identity state. |

### ◆ getKeyRefreshPhase

fsp_err_t(* rm_ble_mesh_access_api_t::getKeyRefreshPhase) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t *const p_key_refresh_state)

To get Key refresh phase.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle identifying the subnet. |
| [out] | p_key_refresh_state | Pointer to key refresh phase state. |

### ◆ setKeyRefreshPhase

fsp_err_t(* rm_ble_mesh_access_api_t::setKeyRefreshPhase) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t const *const p_key_refresh_state)

To set Key refresh phase.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle identifying the subnet. |
| [in] | p_key_refresh_state | Pointer to key refresh phase state. |

### ◆ setTransmitState

fsp_err_t(* rm_ble_mesh_access_api_t::setTransmitState) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t tx_state)

To set Network/Relay Transmit state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | tx_state_type | Transmit state type (Network or Relay). |
| [in] | tx_state | Composite state (3bits of TX count and 5bits of TX interval steps). |

### ◆ getTransmitState

fsp_err_t(* rm_ble_mesh_access_api_t::getTransmitState) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint8_t tx_state_type, uint8_t *const p_tx_state)

To get Network/Relay Transmit state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | tx_state_type | Transmit State Type (Network or Relay). |
| [out] | p_tx_state | Pointer to TX state. |

### ◆ addAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::addAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key)

To add AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the subnet for which AppKey to be added. |
| [in] | appkey_index | Identifies global Index of AppKey. A 12bits value. |
| [in] | p_app_key | Pointer to AppKey. |

### ◆ updateAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::updateAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key)

To update/delete AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the Subnet for which AppKey to be updated/deleted. |
| [in] | appkey_index | Identifies global Index of AppKey. A 12bits value. |
| [in] | p_app_key | Pointer to AppKey. |

### ◆ deleteAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::deleteAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key)

To update/delete AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the Subnet for which AppKey to be updated/deleted. |
| [in] | appkey_index | Identifies global index of AppKey. A 12bits value. |
| [in] | p_app_key | Pointer to AppKey. |

### ◆ getAppKeyHandle

fsp_err_t(* rm_ble_mesh_access_api_t::getAppKeyHandle) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t appkey_index, uint8_t const *const p_app_key, rm_ble_mesh_network_appkey_handle_t *const p_appkey_handle)

To get AppKey Handle for a given AppKey Index.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the Subnet for which AppKey to be gotten. |
| [in] | appkey_index | Identifies global Index of AppKey. A 12bits value. |
| [in] | p_app_key | Pointer to AppKey. |
| [out] | p_app_key_handle | Pointer to AppKey handle. |

### ◆ getAppKeyIndexList

fsp_err_t(* rm_ble_mesh_access_api_t::getAppKeyIndexList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list)

To get list of all known AppKeys.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle of the Subnet for which AppKey to be returned. |
| [out] | p_appkey_count | Pointer to AppKey count. |
| [out] | p_appkey_index_list | Pointer to AppKey index list. |

### ◆ bindModelWithAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::bindModelWithAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index)

To bind a model with an AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Model handle identifying the model. |
| [in] | appkey_index | Identifies global index of AppKey. A 12bits value. |

### ◆ unbindModelWithAppKey

fsp_err_t(* rm_ble_mesh_access_api_t::unbindModelWithAppKey) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t appkey_index)

To unbind a model with an AppKey.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Model handle identifying the model. |
| [in] | appkey_index | Identifies global index of AppKey. A 12bits value. |

### ◆ getModelAppKeyList

fsp_err_t(* rm_ble_mesh_access_api_t::getModelAppKeyList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_appkey_count, uint16_t *const p_appkey_index_list)

To get list of all AppKeys associated with a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which AppKey to be returned. |
| [out] | p_appkey_count | pointer to AppKey count. |
| [out] | p_appkey_index_list | Pointer to AppKey index list. |

### ◆ setModelPublication

fsp_err_t(* rm_ble_mesh_access_api_t::setModelPublication) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info)

To set Publication information associated with a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which Publication info to be set. |
| [in] | p_publish_info | Pointer to publication information structure. |

### ◆ setModelPublicationPeriodDivisor

fsp_err_t(* rm_ble_mesh_access_api_t::setModelPublicationPeriodDivisor) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint8_t period_divisor)

To set Publication Fast Period Divisor information associated with a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which Publication info to be set. |
| [in] | period_divisor | The value range for the Health Fast Period Divisor state is 0 through 15, all other values are prohibited. |

### ◆ getModelPublication

fsp_err_t(* rm_ble_mesh_access_api_t::getModelPublication) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_publish_info_t *const p_publish_info)

To get Publication information associated with a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which Publication info to be returned. |
| [out] | p_publish_info | Pointer to publication information structure. |

### ◆ addModelSubscription

fsp_err_t(* rm_ble_mesh_access_api_t::addModelSubscription) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr)

To add an address to a model subscription list.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which address to be added in the subscription list. |
| [in] | p_sub_addr | Pointer to address structure. |

### ◆ deleteModelSubscription

fsp_err_t(* rm_ble_mesh_access_api_t::deleteModelSubscription) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, rm_ble_mesh_access_address_t const *const p_sub_addr)

To delete an address to a model subscription list.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which address to be deleted in the subscription list. |
| [in] | p_sub_addr | Pointer to address structure. |

### ◆ deleteAllModelSubscription

fsp_err_t(* rm_ble_mesh_access_api_t::deleteAllModelSubscription) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle)

To discard a model subscription list.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which the subscription list to be discarded. |

### ◆ getModelSubscriptionList

fsp_err_t(* rm_ble_mesh_access_api_t::getModelSubscriptionList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t model_handle, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list)

To get list of subscription addresses of a model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | model_handle | Handle of the Model for which the subscription addresses to be returned. |
| [in] | p_sub_addr_count | Pointer to maximum number of subscription address. |
| [out] | p_sub_addr_list | Pointer to subscription addresses. |

### ◆ getAllModelSubscriptionList

fsp_err_t(* rm_ble_mesh_access_api_t::getAllModelSubscriptionList) (rm_ble_mesh_access_ctrl_t *const p_ctrl, uint16_t *const p_sub_addr_count, uint16_t *const p_sub_addr_list)

To get list of subscription addresses of all the models.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_sub_addr_count | Pointer to maximum number of subscription address. |
| [out] | p_sub_addr_list | Pointer to subscription addresses. |

### ◆ isValidElementAddress

fsp_err_t(* rm_ble_mesh_access_api_t::isValidElementAddress) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr)

To check if valid element address to receive a packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | A valid element address, to be checked. |

### ◆ isFixedGroupAddressToBeProcessed

fsp_err_t(* rm_ble_mesh_access_api_t::isFixedGroupAddressToBeProcessed) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr)

To check if Fixed Group Address in receive packet to be processed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | A valid fixed group address, to be checked. |

### ◆ isValidSubscriptionAddress

fsp_err_t(* rm_ble_mesh_access_api_t::isValidSubscriptionAddress) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr)

To check if valid subscription address to receive a packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | A valid subscription address, to be checked. |

### ◆ enableIvUpdateTestMode

fsp_err_t(* rm_ble_mesh_access_api_t::enableIvUpdateTestMode) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_iv_update_test_mode_t mode)

To set the IV Update Test Mode feature.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | mode | This flag is used to either enable or disable the IV update test mode feature. |

### ◆ rm_ble_mesh_access_instance_t

struct rm_ble_mesh_access_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_access_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_access_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_access_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ RM_BLE_MESH_ACCESS_VADDR_LABEL_UUID_SIZE

#define RM_BLE_MESH_ACCESS_VADDR_LABEL_UUID_SIZE

Array sizes for use in the Access layer Size of Virtual Address (Label UUID)

### ◆ RM_BLE_MESH_ACCESS_NETKEY_SIZE

#define RM_BLE_MESH_ACCESS_NETKEY_SIZE

Size of NetKey

### ◆ RM_BLE_MESH_ACCESS_APPKEY_SIZE

#define RM_BLE_MESH_ACCESS_APPKEY_SIZE

Size of AppKey

### ◆ RM_BLE_MESH_ACCESS_KEY_SIZE

| #define RM_BLE_MESH_ACCESS_KEY_SIZE |
| --- |
| Size of Key |

## Typedef Documentation

### ◆ rm_ble_mesh_access_node_id_t

| typedef uint8_t rm_ble_mesh_access_node_id_t |
| --- |
| Access Node ID |

### ◆ rm_ble_mesh_access_element_handle_t

| typedef uint8_t rm_ble_mesh_access_element_handle_t |
| --- |
| Access Element Handle |

### ◆ rm_ble_mesh_access_model_handle_t

| typedef uint16_t rm_ble_mesh_access_model_handle_t |
| --- |
| Access Model Handle |

### ◆ rm_ble_mesh_access_model_id_sig_t

| typedef uint16_t rm_ble_mesh_access_model_id_sig_t |
| --- |
| SIG Model ID |

### ◆ rm_ble_mesh_access_model_id_vendor_t

| typedef uint32_t rm_ble_mesh_access_model_id_vendor_t |
| --- |
| Vendor Model ID |

### ◆ rm_ble_mesh_access_address_handle_t

| typedef uint32_t rm_ble_mesh_access_address_handle_t |
| --- |
| Access Address Handle |

### ◆ rm_ble_mesh_access_device_key_handle_t

| typedef uint32_t rm_ble_mesh_access_device_key_handle_t |
| --- |
| Device Key Handle |

### ◆ rm_ble_mesh_access_ctrl_t

| typedef void rm_ble_mesh_access_ctrl_t |
| --- |
| BLE MESH ACCESS control block. Allocate an instance specific control block to pass into the BLE MESH ACCESS API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_access_model_req_msg_type_t

| enum rm_ble_mesh_access_model_req_msg_type_t | |
| --- | --- |
| Model Specific Request Message Type: Get, Set or Others | |
| Enumerator | |
| RM_BLE_MESH_ACCESS_MODEL_REQ_MSG_TYPE_GET | Model Specific Request Message Type: Get |
| RM_BLE_MESH_ACCESS_MODEL_REQ_MSG_TYPE_SET | Model Specific Request Message Type: Set |
| RM_BLE_MESH_ACCESS_MODEL_REQ_MSG_TYPE_OTHERS | Model Specific Request Message Type: Others |

### ◆ rm_ble_mesh_access_iv_update_test_mode_t

| enum rm_ble_mesh_access_iv_update_test_mode_t | |
| --- | --- |
| Test modes | |
| Enumerator | |
| RM_BLE_MESH_ACCESS_IV_UPDATE_TEST_MODE_DISABLE | Test mode disable. |
| RM_BLE_MESH_ACCESS_IV_UPDATE_TEST_MODE_ENABLE | Test mode enable. |

### ◆ rm_ble_mesh_access_message_opcode_t

| enum rm_ble_mesh_access_message_opcode_t | |
|---|---|
| Opcodes of Model specific messages | |
| Enumerator | |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_APPKEY_DELETE | Config AppKey Delete Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_APPKEY_GET | Config AppKey Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_APPKEY_LIST | Config AppKey List Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_APPKEY_STATUS | Config AppKey Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_ATTENTION_GET | Health Attention Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_ATTENTION_SET | Health Attention Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_ATTENTION_SET_UNACKNOWLEDGED | Health Attention Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_ATTENTION_STATUS | Health Attention Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_COMPOSITION_DATA_GET | Config Composition Data Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_BEACON_GET | Config Beacon Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_BEACON_SET | Config Beacon Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_BEACON_STATUS | Config Beacon Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_DEFAULT_TTL_GET | Config Deafault TTL Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_DEFAULT_TTL_SET | Config Deafault TTL Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_DEFAULT_TTL_STATUS | Config Deafault TTL Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_FRIEND_GET | Config Friend Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_FRIEND_SET | Config Friend Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_FRIEND_STATUS | Config Friend Status Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_GATT_PROXY_GET | Config GATT Proxy Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_GATT_PROXY_SET | Config GATT Proxy Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_GATT_PROXY_STATUS | Config GATT Proxy Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_KEY_REFRESH_PHASE_GET | Config Key Refresh Phase Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_KEY_REFRESH_PHASE_SET | Config Key Refresh Phase Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_KEY_REFRESH_PHASE_STATUS | Config Key Refresh Phase Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_PUBLICATION_GET | Config Model Publication Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_PUBLICATION_STATUS | Config Model Publication Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_PUBLICATION_VIRTUAL_ADDRESS_SET | Config Model Publication Virtual Address Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_ADD | Config Model Subscription Add Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_DELETE | Config Model Subscription Delete Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_DELETE_ALL | Config Model Subscription Delete All Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_OVERWRITE | Config Model Subscription Overwrite Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_STATUS | Config Model Subscription Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_VIRTUAL_ADDRESS_ADD | Config Model Subscription Virtual Address Add Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_VIRTUAL_ADDRESS_DELETE | Config Model Subscription Virtual Address Delete Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_SUBSCRIPTION_VIRTUAL_ADDRESS_OVERWRITE | Config Model Subscription Virtual Address Overwrite Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETWORK_TRANSMIT_GET | Config Network Transmit Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETWORK_TRANSMIT_SET | Config Network Transmit Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CON | |

| | |
|---|---|
| FIG_NETWORK_TRANSMIT_STATUS | Config Network Transmit Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_RELAY_GET | Config Relay Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_RELAY_SET | Config Relay Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_RELAY_STATUS | Config Relay Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_SIG_MODEL_SUBSCRIPTION_GET | Config SIG Model Subscription Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_SIG_MODEL_SUBSCRIPTION_LIST | Config SIG Model Subscription List Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_VENDOR_MODEL_SUBSCRIPTION_GET | Config Vendor Model Subscription Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_VENDOR_MODEL_SUBSCRIPTION_LIST | Config Vendor Model Subscription List Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_LOW_POWER_NODE_POLLTIMEOUT_GET | Config Low Power Node PollTimeout Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_LOW_POWER_NODE_POLLTIMEOUT_STATUS | Config Low Power Node PollTimeout Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_FAULT_CLEAR | Health Fault Clear Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_FAULT_CLEAR_UNACKNOWLEDGED | Health Fault Clear Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_FAULT_GET | Health Fault Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_FAULT_TEST | Health Fault Test Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_FAULT_TEST_UNACKNOWLEDGED | Health Fault Test Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_PERIOD_GET | Health Period Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_PERIOD_SET | Health Period Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_PERIOD_SET_UNACKNOWLEDGED | Health Period Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_HEALTH_PERIOD_STATUS | Health Period Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_HEARTBEAT_PUBLICATION_GET | Config Heartbeat Publication Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_HEARTBEAT_PUBLICATION_SET | Config Heartbeat Publication Set Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_HEARTBEAT_SUBSCRIPTION_GET | Config Heartbeat Subscription Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_HEARTBEAT_SUBSCRIPTION_SET | Config Heartbeat Subscription Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_HEARTBEAT_SUBSCRIPTION_STATUS | Config Heartbeat Subscription Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_APP_BIND | Config Model App Bind Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_APP_STATUS | Config Model App Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_MODEL_APP_UNBIND | Config Model App Unbind Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_ADD | Config NetKey Add Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_DELETE | Config NetKey Delete Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_GET | Config NetKey Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_LIST | Config NetKey List Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_STATUS | Config NetKey Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NETKEY_UPDATE | Config NetKey Update Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NODE_IDENTITY_GET | Config Node Identity Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NODE_IDENTITY_SET | Config Node Identity Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NODE_IDENTITY_STATUS | Config Node Identity Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NODE_RESET | Config Node Reset Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_NODE_RESET_STATUS | Config Node Reset Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_SIG_MODEL_APP_GET | Config SIG Model App Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_SIG_MODEL_APP_LIST | Config SIG Model App List Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_VENDOR_MODEL_APP_GET | Config Vendor Model App Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_CONFIG_VENDOR_MODEL_APP_LIST | Config Vendor Model App List Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONOFF_GET | Generic OnOff Generic OnOff Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONOFF_SET | Generic OnOff Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONOFF_SET_UNACKNOWLEDGED | Generic OnOff Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONOFF_STATUS | Generic OnOff Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LEVEL_GET | Generic Level Generic Level Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LEVEL_SET | Generic Level Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LEVEL_SET_UNACKNOWLEDGED | Generic Level Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LEVEL_STATUS | Generic Level Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DELTA_SET | Generic Delta Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DELTA_SET_UNACKNOWLEDGED | Generic Delta Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MOVE_SET | Generic Move Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MOVE_SET_UNACKNOWLEDGED | Generic Move Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DEFAULT_TRANSITION_TIME_GET | Generic Default Transition Time Generic Default Transition Time Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DEFAULT_TRANSITION_TIME_SET | Generic Default Transition Time Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DEFAULT_TRANSITION_TIME_SET_UNACKNOWLEDGED | Generic Default Transition Time Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_DEFAULT_TRANSITION_TIME_STATUS | Generic Default Transition Time Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONPOWERUP_GET | Generic Power OnOff Generic Power OnOff Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONPOWERUP_STATUS | Generic Power OnOff Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ONPOWERUP_SET | Generic Power OnOff Setup Generic Power OnOff Setup Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GEN | Generic Power OnOff Setup Set |

| | |
|---|---|
| ERIC_ONPOWERUP_SET_UNACKNOWLEDGED | Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LEVEL_GET | Generic Power Level Generic Power Level Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LEVEL_SET | Generic Power Level Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LEVEL_SET_UNACKNOWLEDGED | Generic Power Level Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LEVEL_STATUS | Generic Power Level Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LAST_GET | Generic Power Last Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_LAST_STATUS | Generic Power Last Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_DEFAULT_GET | Generic Power Default Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_DEFAULT_STATUS | Generic Power Default Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_RANGE_GET | Generic Power Range Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_RANGE_STATUS | Generic Power Range Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_DEFAULT_SET | Generic Power Level Setup Generic Power Default Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_DEFAULT_SET_UNACKNOWLEDGED | Generic Power Default Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_RANGE_SET | Generic Power Range Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_POWER_RANGE_SET_UNACKNOWLEDGED | Generic Power Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_BATTERY_GET | Generic Battery Generic Battery Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_BATTERY_STATUS | Generic Battery Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_GLOBAL_GET | Generic Location Generic Location Global Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_GLOBAL_STATUS | Generic Location Global Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GEN | Generic Location Local Get Opcode |

| | |
|---|---|
| ERIC_LOCATION_LOCAL_GET | |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_LOCAL_STATUS | Generic Location Local Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_GLOBAL_SET | Generic Location Setup Generic Location Global Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_GLOBAL_SET_UNACKNOWLEDGED | Generic Location Global Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_LOCAL_SET | Generic Location Local Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_LOCATION_LOCAL_SET_UNACKNOWLEDGED | Generic Location Local Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTIES_GET | Generic Manufacturer Property Generic Manufacturer Properties Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTIES_STATUS | Generic Manufacturer Properties Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTY_GET | Generic Manufacturer Property Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTY_SET | Generic Manufacturer Property Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTY_SET_UNACKNOWLEDGED | Generic Manufacturer Property Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_MANUFACTURER_PROPERTY_STATUS | Generic Manufacturer Property Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTIES_GET | Generic Admin Property Generic Admin Properties Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTIES_STATUS | Generic Admin Properties Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTY_GET | Generic Admin Property Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTY_SET | Generic Admin Property Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTY_SET_UNACKNOWLEDGED | Generic Admin Property Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_ADMIN_PROPERTY_STATUS | Generic Admin Property Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTIES_GET | Generic User Property Generic User Properties Get Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTIES_STATUS | Generic User Properties Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTY_GET | Generic User Property Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTY_SET | Generic User Property Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTY_SET_UNACKNOWLEDGED | Generic User Property Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_USER_PROPERTY_STATUS | Generic User Property Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_CLIENT_PROPERTIES_GET | Generic Client Property Generic Client Properties Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_GENERIC_CLIENT_PROPERTIES_STATUS | Generic Client Properties Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_DESCRIPTOR_GET | Sensor Sensor Descriptor Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_DESCRIPTOR_STATUS | Sensor Descriptor Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_GET | Sensor Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_STATUS | Sensor Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_COLUMN_GET | Sensor Column Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_COLUMN_STATUS | Sensor Column Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SERIES_GET | Sensor Series Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SERIES_STATUS | Sensor Series Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_CADENCE_GET | Sensor Setup Sensor Cadence Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_CADENCE_SET | Sensor Cadence Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_CADENCE_SET_UNACKNOWLEDGED | Sensor Cadence Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_CADENCE_STATUS | Sensor Cadence Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SETTINGS_GET | Sensor Settings Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SEN | |

| | |
|---|---|
| SOR_SETTINGS_STATUS | Sensor Settings Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SETTING_GET | Sensor Setting Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SETTING_SET | Sensor Setting Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SETTING_SET_UNACKNOWLEDGED | Sensor Setting Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SENSOR_SETTING_STATUS | Sensor Setting Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_GET | Time Time Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_SET | Time Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_STATUS | Time Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ROLE_GET | Time Role Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ROLE_SET | Time Role Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ROLE_STATUS | Time Role Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ZONE_GET | Time Zone Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ZONE_SET | Time Zone Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TIME_ZONE_STATUS | Time Zone Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TAI_UTC_DELTA_GET | Time - TAI UTC Delta Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TAI_UTC_DELTA_SET | Time - TAI UTC Delta Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_TAI_UTC_DELTA_STATUS | Time - TAI UTC Delta Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_GET | Scene Scene Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_RECALL | Scene Recall Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_RECALL_UNACKNOWLEDGED | Scene Recall Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_STATUS | Scene Status Opcode |
| | |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_REGISTER_GET | Scene Register Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_REGISTER_STATUS | Scene Register Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_STORE | Scene Setup Scene Store Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_STORE_UNACKNOWLEDGED | Scene Store Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_DELETE | Scene Delete Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCENE_DELETE_UNACKNOWLEDGED | Scene Delete Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_ACTION_GET | Scheduler Scheduler Action Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_ACTION_STATUS | Scheduler Action Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_GET | Scheduler Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_STATUS | Scheduler Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_ACTION_SET | Scheduler Setup Scheduler Action Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_SCHEDULER_ACTION_SET_UNACKNOWLEDGED | Scheduler Action Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_GET | Light Lightness Light Lightness Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_SET | Light Lightness Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_SET_UNACKNOWLEDGED | Light Lightness Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_STATUS | Light Lightness Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_LINEAR_GET | Light Lightness Linear Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_LINEAR_SET | Light Lightness Linear Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_LINEAR_SET_UNACKNOWLEDGED | Light Lightness Linear Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_LINEAR_STATUS | Light Lightness Linear Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIG | Light Lightness Last Get Opcode |

| | |
|---|---|
| HT_LIGHTNESS_LAST_GET | |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_LAST_STATUS | Light Lightness Last Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_DEFAULT_GET | Light Lightness Default Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_DEFAULT_STATUS | Light Lightness Default Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_RANGE_GET | Light Lightness Range Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_RANGE_STATUS | Light Lightness Range Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_DEFAULT_SET | Light Lightness Setup Light Lightness Range Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_DEFAULT_SET_UNACKNOWLEDGED | Light Lightness Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_RANGE_SET | Light Lightness Range Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LIGHTNESS_RANGE_SET_UNACKNOWLEDGED | Light Lightness Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_GET | Light CTL Light CTL Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_SET | Light CTL Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_SET_UNACKNOWLEDGED | Light CTL Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_STATUS | Light CTL Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_GET | Light CTL Temperature Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_RANGE_GET | Light CTL Temperature Range Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_RANGE_STATUS | Light CTL Temperature Range Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_SET | Light CTL Temperature Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_SET_UNACKNOWLEDGED | Light CTL Temperature Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_STATUS | Light CTL Temperature Status Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_DEFAULT_GET | Light CTL Default Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_DEFAULT_STATUS | Light CTL Default Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_DEFAULT_SET | Light CTL Setup Light CTL Default Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_DEFAULT_SET_UNACKNOWLEDGED | Light CTL Default Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_RANGE_SET | Light CTL Default Range Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNACKNOWLEDGED | Light CTL Default Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_GET | Light HSL Light HSL Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_HUE_GET | Light HSL HUE Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_HUE_SET | Light HSL HUE Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_HUE_SET_UNACKNOWLEDGED | Light HSL HUE Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_HUE_STATUS | Light HSL HUE Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SATURATION_GET | Light HSL Saturation Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SATURATION_SET | Light HSL Saturation Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SATURATION_SET_UNACKNOWLEDGED | Light HSL Saturation Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SATURATION_STATUS | Light HSL Saturation Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SET | Light HSL Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_SET_UNACKNOWLEDGED | Light HSL Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_STATUS | Light HSL Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_TARGET_GET | Light HSL Target Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_TARGET_STATUS | Light HSL Target Status Opcode |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_DEFAULT_GET | Light HSL Default Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_DEFAULT_STATUS | Light HSL Default Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_RANGE_GET | Light HSL Range Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_RANGE_STATUS | Light HSL Range Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_DEFAULT_SET | Light HSL Setup Light HSL Default Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_DEFAULT_SET_UNACKNOWLEDGED | Light HSL Default Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_RANGE_SET | Light HSL Range Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_HSL_RANGE_SET_UNACKNOWLEDGED | Light HSL Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_GET | Light xyL Light xyL Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_SET | Light xyL Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_SET_UNACKNOWLEDGED | Light xyL Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_STATUS | Light xyL Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_TARGET_GET | Light xyL Target Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_TARGET_STATUS | Light xyL Target Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_DEFAULT_GET | Light xyL Default Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_DEFAULT_STATUS | Light xyL Default Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_RANGE_GET | Light xyL Range Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_RANGE_STATUS | Light xyL Range Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_DEFAULT_SET | Light xyL Setup Light xyL Default Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_DEFAULT_SET_UNACKNOWLEDGED | Light xyL Default Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIG | Light xyL Range Set Opcode |

| | |
|---|---|
| HT_XYL_RANGE_SET | |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_XYL_RANGE_SET_UNACKNOWLEDGED | Light xyL Range Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_MODE_GET | Light Control Light LC Mode Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_MODE_SET | Light LC Mode Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_MODE_SET_UNACKNOWLEDGED | Light LC Mode Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_MODE_STATUS | Light LC Mode Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_OM_GET | Light LC Occupancy Mode Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_OM_SET | Light LC Occupancy Mode Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_OM_SET_UNACKNOWLEDGED | Light LC Occupancy Mode Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_OM_STATUS | Light LC Occupancy Mode Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_LIGHT_ONOFF_GET | Light LC Light OnOff Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_LIGHT_ONOFF_SET | Light LC Light OnOff Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_LIGHT_ONOFF_SET_UNACKNOWLEDGED | Light LC Light OnOff Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_LIGHT_ONOFF_STATUS | Light LC Light OnOff Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_PROPERTY_GET | Light LC Property Get Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_PROPERTY_SET | Light LC Property Set Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_PROPERTY_SET_UNACKNOWLEDGED | Light LC Property Set Unacknowledged Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_LIGHT_LC_PROPERTY_STATUS | Light LC Property Status Opcode |
| RM_BLE_MESH_ACCESS_MESSAGE_OPCODE_INVALID | Invalid Opcode |

### ◆ rm_ble_mesh_access_model_type_info_t

| enum rm_ble_mesh_access_model_type_info_t | |
|---|---|
| Model type information | |
| Enumerator | |
| RM_BLE_MESH_ACCESS_MODEL_TYPE_INFO_SIG | Model type - SIG |
| RM_BLE_MESH_ACCESS_MODEL_TYPE_INFO_VENDOR | Model type - VENDOR |

### ◆ rm_ble_mesh_access_model_id_info_t

| enum rm_ble_mesh_access_model_id_info_t | |
|---|---|
| Model ID information | |
| Enumerator | |
| RM_BLE_MESH_ACCESS_MODEL_INFO_CONFIG_SERVER | Model ID - Config Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_CONFIG_CLIENT | Model ID - Config Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_HEALTH_SERVER | Model ID - Health Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_HEALTH_CLIENT | Model ID - Health Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_ONOFF_SERVER | Generic Model ID - Generic OnOff Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_ONOFF_CLIENT | Model ID - Generic OnOff Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_LEVEL_SERVER | Model ID - Generic Level Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_LEVEL_CLIENT | Model ID - Generic Level Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_DEFAULT_TRANSITION_TIME_SERVER | Model ID - Generic Default Transition Time Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_DEFAULT_TRANSITION_TIME_CLIENT | Model ID - Generic Default Transition Time Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_ONOFF_SERVER | Model ID - Generic Power OnOff Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_ONOFF_SETUP_SERVER | Model ID - Generic Power OnOff Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_ONOFF_CLIENT | Model ID - Generic Power OnOff Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_LEVEL_SERVER | Model ID - Generic Power Level Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_LEVEL_SETUP_SERVER | Model ID - Generic Power Level Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_POWER_LEVEL_CLIENT | Model ID - Generic Power Level Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_BATTERY_SERVER | Model ID - Generic Battery Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_ | |

| | |
|---|---|
| BATTERY_CLIENT | Model ID - Generic Battery Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_LOCATION_SERVER | Model ID - Generic Location Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_LOCATION_SETUP_SERVER | Model ID - Generic Location Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_LOCATION_CLIENT | Model ID - Generic Location Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_ADMIN_PROPERTY_SERVER | Model ID - Generic Admin Property Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_MANUFACTURER_PROPERTY_SERVER | Model ID - Generic Manufacturer Property Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_USER_PROPERTY_SERVER | Model ID - Generic User Property Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_CLIENT_PROPERTY_SERVER | Model ID - Generic Client Property Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_GENERIC_PROPERTY_CLIENT | Model ID - Generic Property Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SENSOR_SERVER | Sensors Model ID - Sensor Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SENSOR_SETUP_SERVER | Model ID - Sensor Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SENSOR_CLIENT | Model ID - Sensor Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_TIME_SERVER | Time and Scenes Model ID - Time Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_TIME_SETUP_SERVER | Model ID - Time Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_TIME_CLIENT | Model ID - Time Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCENE_SERVER | Model ID - Scene Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCENE_SETUP_SERVER | Model ID - Scene Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCENE_CLIENT | Model ID - Scene Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCHEDULER_SERVER | Model ID - Scheduler Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCHEDULER_SETUP_SERVER | Model ID - Scheduler Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_SCHEDULER_CLIENT | Model ID - Scheduler Client |

| | |
|---|---|
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LIGHTNESS_SERVER | Lighting Model ID - Light Lightness Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LIGHTNESS_SETUP_SERVER | Model ID - Light Lightness Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LIGHTNESS_CLIENT | Model ID - Light Lightness Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_CTL_SERVER | Model ID - Light CTL Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_CTL_SETUP_SERVER | Model ID - Light CTL Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_CTL_CLIENT | Model ID - Light CTL Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_CTL_TEMPERATURE_SERVER | Model ID - Light CTL Temperature Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_HSL_SERVER | Model ID - Light HSL Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_HSL_SETUP_SERVER | Model ID - Light HSL Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_HSL_CLIENT | Model ID - Light HSL Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_HSL_HUE_SERVER | Model ID - Light HSL HUE Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_HSL_SATURATION_SERVER | Model ID - Light HSL Saturation Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_XYL_SERVER | Model ID - Light xyL Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_XYL_SETUP_SERVER | Model ID - Light xyL Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_XYL_CLIENT | Model ID - Light xyL Client |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LC_SERVER | Model ID - Light LC Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LC_SETUP_SERVER | Model ID - Light LC Setup Server |
| RM_BLE_MESH_ACCESS_MODEL_INFO_LIGHT_LC_CLIENT | Model ID - Light LC Client |

### Application Callback

Interfaces » Networking » BLE Mesh Network Interfaces » BLE Mesh Access Interface

## Detailed Description

This Section Describes the module Notification Callback interface offered to the application

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_access_model_callback_args_t |
| struct | rm_ble_mesh_access_timeout_callback_args_t |

### Typedefs

| | |
|---|---|
| typedef void(* | rm_ble_mesh_access_model_callback_t)<br>(rm_ble_mesh_access_model_callback_args_t *p_args) |
| typedef void(* | rm_ble_mesh_access_timeout_callback_t)<br>(rm_ble_mesh_access_timeout_callback_args_t *p_args) |

## Data Structure Documentation

### ◆ rm_ble_mesh_access_model_callback_args_t

struct rm_ble_mesh_access_model_callback_args_t

Access Layer Application Asynchronous Notification Callback Arguments.

Data Fields

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_req_msg_context_t * | p_msg_context | Context of message received for a specific model instance. |
| rm_ble_mesh_access_req_msg_raw_t * | p_msg_raw | Uninterpreted/raw received message for a specific model instance. |

### ◆ rm_ble_mesh_access_timeout_callback_args_t

struct rm_ble_mesh_access_timeout_callback_args_t

Access Layer Model Publication Timeout Callback Arguments.

Access Layer calls the registered callback to indicate Publication Timeout for the associated model.

**Parameters**

| | |
|---|---|
| p_context | Placeholder for user data. |
| handle | Model Handle. |
| blob | Blob if any or NULL. |

Data Fields

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_ha | handle | Model handle. |

| | | |
|---|---|---|
| ndle_t | | |
| void * | p_blob | Blob if any or NULL. |

### Typedef Documentation

#### ◆ rm_ble_mesh_access_model_callback_t

typedef void(* rm_ble_mesh_access_model_callback_t)
(rm_ble_mesh_access_model_callback_args_t *p_args)

Access Layer Application Asynchronous Notification Callback.

Access Layer calls the registered callback to indicate events occurred to the application.

**Parameters**

| | |
|---|---|
| p_args | Access Layer application asynchronous notification callback arguments. |

#### ◆ rm_ble_mesh_access_timeout_callback_t

typedef void(* rm_ble_mesh_access_timeout_callback_t)
(rm_ble_mesh_access_timeout_callback_args_t *p_args)

Access Layer Model Publication Timeout Callback.

Access Layer calls the registered callback to indicate Publication Timeout for the associated model.

**Parameters**

| | |
|---|---|
| p_args | Access Layer Model publication timeout callback arguments. |

## BLE Mesh Bearer Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Bearer functions.

# Summary

The BLE Mesh Bearer interface for the BLE Mesh Bearer (BLE MESH BEARER) peripheral provides BLE Mesh Bearer functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_bearer_info_t |
| struct | rm_ble_mesh_bearer_beacon_info_t |
| struct | rm_ble_mesh_bearer_ch_info_t |
| struct | rm_ble_mesh_bearer_ntf_callback_args_t |
| struct | rm_ble_mesh_bearer_beacon_callback_args_t |
| struct | rm_ble_mesh_bearer_cfg_t |
| struct | rm_ble_mesh_bearer_api_t |
| struct | rm_ble_mesh_bearer_instance_t |

### Macros

| | |
|---|---|
| #define | RM_BLE_MESH_BEARER_HANDLE_INVALID |
| #define | RM_BLE_MESH_BEARER_MAX_PDU_SIZE |
| #define | RM_BLE_MESH_BEARER_SUBTYPE_GATT_T_MASK_BIT_OFFSET |

### Typedefs

| | |
|---|---|
| typedef uint8_t | rm_ble_mesh_bearer_handle_t |
| typedef void | rm_ble_mesh_bearer_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_bearer_interface_event_t |
| enum | rm_ble_mesh_bearer_beacon_operation_t |
| enum | rm_ble_mesh_bearer_beacon_action_t |
| enum | rm_ble_mesh_bearer_role_t |
| enum | rm_ble_mesh_bearer_operation_mode_t |
| enum | rm_ble_mesh_bearer_ntf_callback_result_t |
| enum | rm_ble_mesh_bearer_type_t |
| enum | rm_ble_mesh_bearer_beacon_type_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_bearer_info_t

| struct rm_ble_mesh_bearer_info_t | |
|---|---|
| Bearer information to register | |
| **Data Fields** | |
| rm_ble_mesh_buffer_t * | binfo |
| | |
| fsp_err_t(* | bearer_send )(rm_ble_mesh_bearer_handle_t *, uint8_t, void *, uint16_t) |
| | |
| void(* | bearer_recv )(rm_ble_mesh_bearer_handle_t *, uint8_t *, uint16_t, rm_ble_mesh_buffer_t *info) |
| | |
| void(* | bearer_sleep )(rm_ble_mesh_bearer_handle_t *) |
| | |
| void(* | bearer_wakeup )(rm_ble_mesh_bearer_handle_t *, uint8_t mode) |
| | |

## Field Documentation

### ◆ binfo

rm_ble_mesh_buffer_t* rm_ble_mesh_bearer_info_t::binfo

Bearer Information

### ◆ bearer_send

fsp_err_t(* rm_ble_mesh_bearer_info_t::bearer_send) (rm_ble_mesh_bearer_handle_t *, uint8_t, void *, uint16_t)

Data Send routine

### ◆ bearer_recv

void(* rm_ble_mesh_bearer_info_t::bearer_recv) (rm_ble_mesh_bearer_handle_t *, uint8_t *, uint16_t, rm_ble_mesh_buffer_t *info)

Data Receive routine

### ◆ bearer_sleep

void(* rm_ble_mesh_bearer_info_t::bearer_sleep) (rm_ble_mesh_bearer_handle_t *)

Bearer Sleep Interface

### ◆ bearer_wakeup

| void(* rm_ble_mesh_bearer_info_t::bearer_wakeup) (rm_ble_mesh_bearer_handle_t *, uint8_t mode) |
|---|
| Bearer Wakeup Interface |

### ◆ rm_ble_mesh_bearer_beacon_info_t

| struct rm_ble_mesh_bearer_beacon_info_t | | |
|---|---|---|
| Bearer Beacon type data structure | | |
| Data Fields | | |
| uint8_t | action | Beacon Action<br>• Lower Nibble:<br>BRR_OBSERVE BRR_BROADCAST<br>• Higher Nibble:<br>BRR_ENABLE BRR_DISABLE |
| uint8_t | type | Beacon type<br>• Lower Nibble:<br>BRR_BCON_PASSIVE - Non Connectable beacon BRR_BCON_ACTIVE - Connectable beacon<br>• Higher Nibble (Valid only when Passive)<br>BRR_BCON_TYPE_UNPROV_DEVICE BRR_BCON_TYPE_SECURE_NET |
| uint8_t * | bcon_data | Beacon Broadcast Data |

| | | |
|---|---|---|
| uint16_t | bcon_datalen | Beacon Broadcast Data length |
| rm_ble_mesh_buffer_t * | uri | URI information in case of Unprovisioned Beacons |

### ◆ rm_ble_mesh_bearer_ch_info_t

| struct rm_ble_mesh_bearer_ch_info_t | | |
|---|---|---|
| Bearer GATT Channel information related data structure | | |
| Data Fields | | |
| uint16_t | mtu | Identifies the MTU for the Bearer Channel |
| uint8_t | role | Identifies the role for the Bearer channel |

### ◆ rm_ble_mesh_bearer_ntf_callback_args_t

| struct rm_ble_mesh_bearer_ntf_callback_args_t | | |
|---|---|---|
| BEARER Application Asynchronous Notification Callback.<br>BEARER calls the registered callback to indicate events occurred to the application. | | |
| Data Fields | | |
| rm_ble_mesh_bearer_handle_t * | p_handle | Bearer handle identifier. |
| rm_ble_mesh_bearer_interface_event_t | event | Bearer interface event. |
| uint8_t * | p_data | Data associated with the event if any or NULL. |
| uint16_t | data_length | Size of the event data. 0 if event data is NULL. |

### ◆ rm_ble_mesh_bearer_beacon_callback_args_t

| struct rm_ble_mesh_bearer_beacon_callback_args_t | | |
|---|---|---|
| BEARER Application Asynchronous Notification Callback for Beacons.<br>Application registers callback for beacon notification with bearer. | | |
| Data Fields | | |
| uint8_t * | p_data | Data associated with the event if any or NULL. |
| uint16_t | data_length | Size of the event data. 0 if event data is NULL. |

### ◆ rm_ble_mesh_bearer_cfg_t

| struct rm_ble_mesh_bearer_cfg_t |
|---|
| BLE MESH BEARER configuration parameters. |
| Data Fields |

| | | |
|---|---|---|
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware.<br><br>the parameters for initialization. |
| rm_ble_mesh_instance_t const * | p_mesh_instance | Instance structure of BLE Mesh. |
| void * | p_context | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| void const * | p_extend | Placeholder for user extension. |

### ◆ rm_ble_mesh_bearer_api_t

| struct rm_ble_mesh_bearer_api_t | |
|---|---|
| BLE MESH BEARER functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | registerInterface )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_ntf_callback_result_t (*p_callback)(rm_ble_mesh_bearer_ntf_callback_args_t *p_args)) |
| | |
| fsp_err_t(* | registerBeaconHandler )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_beacon_type_t bcon_type, void(*p_handler)(rm_ble_mesh_bearer_beacon_callback_args_t *p_args)) |
| | |
| fsp_err_t(* | addBearer )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_info_t const *const p_brr_info, rm_ble_mesh_bearer_handle_t *const p_brr_handle) |
| | |
| fsp_err_t(* | removeBearer )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_handle_t const *const p_brr_handle) |
| | |
| fsp_err_t(* | observeBeacon )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t bcon_type, uint8_t enable) |

| | |
|---|---|
| fsp_err_t(* | bcastUnprovisionedBeacon )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_dev_uuid, uint16_t oob_info, rm_ble_mesh_buffer_t const *const p_uri) |
| fsp_err_t(* | broadcastBeacon )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_packet, uint16_t length) |
| fsp_err_t(* | startProxyAdv )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_data, uint16_t datalen) |
| fsp_err_t(* | sendPdu )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_handle_t const *const p_brr_handle, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_buffer_t const *const p_buffer) |
| fsp_err_t(* | getPacketRssi )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t *p_rssi_value) |
| fsp_err_t(* | sleep )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl) |
| fsp_err_t(* | wakeup )(rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t mode) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_bearer_api_t::open) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_cfg_t const *const p_cfg)

Open bearer middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_bearer_api_t::close) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl)

Close bearer middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ registerInterface

fsp_err_t(* rm_ble_mesh_bearer_api_t::registerInterface) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_ntf_callback_result_t(*p_callback)( rm_ble_mesh_bearer_ntf_callback_args_t *p_args))

Register Interface with Bearer Layer

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | brr_type | Bearer Type. |
| [in] | p_callback | Pointer to details describing application notification callback. |

### ◆ registerBeaconHandler

fsp_err_t(* rm_ble_mesh_bearer_api_t::registerBeaconHandler) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_beacon_type_t bcon_type, void( *p_handler)( rm_ble_mesh_bearer_beacon_callback_args_t *p_args))

Register Beacon Interface with Bearer Layer

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | bcon_type | Beacon type - Unprovisioned Device or Secure Network. |
| [in] | p_handler | Pointer to callback handler to be registered for the given beacon type. |

### ◆ addBearer

fsp_err_t(* rm_ble_mesh_bearer_api_t::addBearer) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_info_t const *const p_brr_info, rm_ble_mesh_bearer_handle_t *const p_brr_handle)

Add a bearer to Bearer Layer

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | brr_type | Bearer Type. |
| [in] | p_brr_info | Pointer to details describing the Bearer being added. |
| [out] | p_brr_handle | Pointer to handle to the bearer that is added. Used in data APIs. |

### ◆ removeBearer

fsp_err_t(* rm_ble_mesh_bearer_api_t::removeBearer) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_bearer_handle_t const *const p_brr_handle)

Remove a bearer from Bearer Layer

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | brr_type | Bearer Type. |
| [in] | p_brr_handle | Pointer to handle to the bearer is removed. Used in data APIs. |

### ◆ observeBeacon

fsp_err_t(* rm_ble_mesh_bearer_api_t::observeBeacon) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t bcon_type, uint8_t enable)

Observe ON/OFF for the beacon type

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | bcon_type | Type of beacon to observe - Active/Passive. |
| [in] | enable | Enable or Disable the observation procedure. |

### ◆ bcastUnprovisionedBeacon

fsp_err_t(* rm_ble_mesh_bearer_api_t::bcastUnprovisionedBeacon) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_dev_uuid, uint16_t oob_info, rm_ble_mesh_buffer_t const *const p_uri)

API to send Unprovisioned Device Beacon

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | type | Active or Passive Broadcast type. |
| [in] | p_dev_uuid | Pointer to device UUID uniquely identifying this device. |
| [in] | oob_info | OOB Information. |
| [in] | p_uri | Pointer to optional Parameter. NULL if not present. Points to the length and payload pointer of the URI string to be advertised interleaving with the unprovisioned beacon. |

### ◆ broadcastBeacon

fsp_err_t(* rm_ble_mesh_bearer_api_t::broadcastBeacon) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_packet, uint16_t length)

API to broadcast a beacon

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | type | The type of beacon. |
| [in] | p_packet | Pointer to beacon data. |
| [in] | length | Beacon data length. |

### ◆ startProxyAdv

fsp_err_t(* rm_ble_mesh_bearer_api_t::startProxyAdv) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t type, uint8_t const *const p_data, uint16_t datalen)

API to send Proxy Device ADV

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | type | Proxy ADV Type: 0x00 - Network ID 0x01 - Node Identity |
| [in] | p_data | Pointer to data to be advertised by Proxy. If the "type" is: 0x00 - Network ID - 8 Bytes of Network ID 0x01 - Node Identity - 8 Bytes Hash, 8 Bytes Random number |
| [in] | datalen | Length of the data to be advertised by Proxy. If the "type" is: 0x00 - Network ID - 8 Bytes of Network ID 0x01 - Node Identity - 8 Bytes Hash, 8 Bytes Random number |

### ◆ sendPdu

fsp_err_t(* rm_ble_mesh_bearer_api_t::sendPdu) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, rm_ble_mesh_bearer_handle_t const *const p_brr_handle, rm_ble_mesh_bearer_type_t brr_type, rm_ble_mesh_buffer_t const *const p_buffer)

Send a bearer PDU

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_brr_handle | Pointer to bearer handle on which PDU is to be sent. |
| [in] | brr_type | Bearer Type. |
| [in] | p_buffer | Pointer to PDU data to be sent. |

### ◆ getPacketRssi

fsp_err_t(* rm_ble_mesh_bearer_api_t::getPacketRssi) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t *p_rssi_value)

Get the RSSI of current received packet being processed.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_rssi_value | Pointer to RSSI value. |

***Note***

*This applies only when the packet is received over ADV bearer.*

### ◆ sleep

fsp_err_t(* rm_ble_mesh_bearer_api_t::sleep) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl)

Put the bearer to sleep.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ wakeup

fsp_err_t(* rm_ble_mesh_bearer_api_t::wakeup) (rm_ble_mesh_bearer_ctrl_t *const p_ctrl, uint8_t mode)

Wakeup the bearer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | mode | Identifies the mode (BRR_TX/BRR_RX) for which bearer is requested for wakeup. |

### ◆ rm_ble_mesh_bearer_instance_t

struct rm_ble_mesh_bearer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_bearer_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_bearer_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_bearer_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ RM_BLE_MESH_BEARER_HANDLE_INVALID

| #define RM_BLE_MESH_BEARER_HANDLE_INVALID |
| --- |
| Invalid Bearer handle identifier |

### ◆ RM_BLE_MESH_BEARER_MAX_PDU_SIZE

| #define RM_BLE_MESH_BEARER_MAX_PDU_SIZE |
| --- |
| Maximum PDU size for data received over bearer |

### ◆ RM_BLE_MESH_BEARER_SUBTYPE_GATT_T_MASK_BIT_OFFSET

| #define RM_BLE_MESH_BEARER_SUBTYPE_GATT_T_MASK_BIT_OFFSET |
| --- |
| GATT Bearer Message Type Masks |

## Typedef Documentation

### ◆ rm_ble_mesh_bearer_handle_t

| typedef uint8_t rm_ble_mesh_bearer_handle_t |
| --- |
| Bearer handle identifier |

### ◆ rm_ble_mesh_bearer_ctrl_t

| typedef void rm_ble_mesh_bearer_ctrl_t |
| --- |
| BLE MESH BEARER control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_bearer_interface_event_t

| enum rm_ble_mesh_bearer_interface_event_t | |
|---|---|
| Bearer Interface Event | |
| Enumerator | |
| RM_BLE_MESH_BEARER_INTERFACE_EVENT_DOWN | Bearer Interface Event - Down |
| RM_BLE_MESH_BEARER_INTERFACE_EVENT_UP | Bearer Interface Event - Up |
| RM_BLE_MESH_BEARER_INTERFACE_EVENT_DATA | Bearer Interface Event - Data |
| RM_BLE_MESH_BEARER_INTERFACE_EVENT_PROXY_DATA | Bearer Interface Event - Proxy Data |

### ◆ rm_ble_mesh_bearer_beacon_operation_t

| enum rm_ble_mesh_bearer_beacon_operation_t | |
|---|---|
| Bearer Beacon Operations | |
| Enumerator | |
| RM_BLE_MESH_BEARER_BEACON_OPERATION_BROADCAST | Bearer Beacon Operations - Broadcast |
| RM_BLE_MESH_BEARER_BEACON_OPERATION_OBSERVE | Bearer Beacon Operations - Observe |

### ◆ rm_ble_mesh_bearer_beacon_action_t

| enum rm_ble_mesh_bearer_beacon_action_t | |
|---|---|
| Bearer Beacon Actions | |
| Enumerator | |
| RM_BLE_MESH_BEARER_BEACON_ACTION_DISABLE | Bearer Beacon Actions - Disable |
| RM_BLE_MESH_BEARER_BEACON_ACTION_ENABLE | Bearer Beacon Actions - Enable |

### ◆ rm_ble_mesh_bearer_role_t

| enum rm_ble_mesh_bearer_role_t | |
|---|---|
| Bearer Server Client Roles | |
| Enumerator | |
| RM_BLE_MESH_BEARER_ROLE_CLIENT | Bearer Client Role |
| RM_BLE_MESH_BEARER_ROLE_SERVER | Bearer Server Role |
| RM_BLE_MESH_BEARER_ROLE_INVALID | Bearer Role - Invalid |

### ◆ rm_ble_mesh_bearer_operation_mode_t

| enum rm_ble_mesh_bearer_operation_mode_t | |
|---|---|
| Bearer Transmit and Receive operation modes | |
| Enumerator | |
| RM_BLE_MESH_BEARER_OPERATION_MODE_TX | Bearer transmit operation mode |
| RM_BLE_MESH_BEARER_OPERATION_MODE_RX | Bearer receive operation mode |

### ◆ rm_ble_mesh_bearer_ntf_callback_result_t

| enum rm_ble_mesh_bearer_ntf_callback_result_t | |
|---|---|
| Bearer Transmit and Receive operation modes | |
| Enumerator | |
| RM_BLE_MESH_BEARER_NTF_CALLBACK_RESULT_SUCCESS | Callback success |
| RM_BLE_MESH_BEARER_NTF_CALLBACK_RESULT_FAILURE | Callback failure |

### ◆ rm_ble_mesh_bearer_type_t

| enum rm_ble_mesh_bearer_type_t | |
|---|---|
| Bearer Type definitions | |
| Enumerator | |
| RM_BLE_MESH_BEARER_TYPE_BCON | Beacon Bearer |
| RM_BLE_MESH_BEARER_TYPE_ADV | Advertising Bearer |
| RM_BLE_MESH_BEARER_TYPE_PB_ADV | Provisioning Advertising Bearer |
| RM_BLE_MESH_BEARER_TYPE_GATT | GATT Bearer |
| RM_BLE_MESH_BEARER_TYPE_PB_GATT | Provisioning GATT Bearer |
| RM_BLE_MESH_BEARER_COUNT | Number of bearers supported |

### ◆ rm_ble_mesh_bearer_beacon_type_t

| enum rm_ble_mesh_bearer_beacon_type_t | |
|---|---|
| Bearer Beacon type definitions | |
| Enumerator | |
| RM_BLE_MESH_BEARER_BEACON_TYPE_UNPROV_DEVICE | Unprovisioned Device Beacon |
| RM_BLE_MESH_BEARER_BEACON_TYPE_SECURE_NET | Secure Network Beacon |
| RM_BLE_MESH_BEARER_BEACON_TYPE_GATT_UNPROV_DEVICE | Unprovisioned Device Beacon over GATT bearer |
| RM_BLE_MESH_BEARER_BEACON_TYPE_PROXY_NETID | Proxy beacon with Network ID |
| RM_BLE_MESH_BEARER_BEACON_TYPE_PROXY_NODEID | Proxy beacon with Node Identity |
| RM_BLE_MESH_BEARER_BEACON_COUNT | Number of Beacon types |

**BLE Mesh Bearer Platform Interface**
Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Bearer Platform functions.

# Summary

The BLE Mesh interface for the BLE Mesh Bearer Platform (BLE MESH BEARER PLATFORM) peripheral provides BLE Mesh Bearer Platform functionality.

### Data Structures

| | |
|---|---|
| struct | rm_mesh_bearer_platform_cfg_t |
| struct | rm_mesh_bearer_platform_api_t |
| struct | rm_mesh_bearer_platform_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_mesh_bearer_platform_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_mesh_bearer_platform_device_address_type_t |
| enum | rm_mesh_bearer_platform_state_t |
| enum | rm_mesh_bearer_platform_gatt_mode_t |

## Data Structure Documentation

### ◆ rm_mesh_bearer_platform_cfg_t

| struct rm_mesh_bearer_platform_cfg_t | | |
|---|---|---|
| MESH BEARER PLATFORM configuration parameters. | | |
| Data Fields | | |
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware.<br><br>the parameters for initialization. |
| rm_mesh_bearer_platform_device_address_type_t | device_address_type | Device address type. |
| rm_ble_mesh_bearer_instance_t const * | p_bearer_instance | Instance structure of BLE Mesh Bearer. |
| void * | p_context | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |

| | | |
|---|---|---|
| void const * | p_extend | Placeholder for user extension. |

### ◆ rm_mesh_bearer_platform_api_t

| struct rm_mesh_bearer_platform_api_t | |
|---|---|
| MESH BEARER PLATFORM functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | setScanResponseData )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_data, uint8_t len) |
| | |
| fsp_err_t(* | connect )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_remote_address, uint8_t address_type, rm_mesh_bearer_platform_gatt_mode_t mode) |
| | |
| fsp_err_t(* | discoverService )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_gatt_mode_t mode) |
| | |
| fsp_err_t(* | configureNotification )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_state_t state, rm_mesh_bearer_platform_gatt_mode_t mode) |
| | |
| fsp_err_t(* | disconnect )(rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_mesh_bearer_platform_api_t::open) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, rm_mesh_bearer_platform_cfg_t const *const p_cfg)

Open Bearer Platform middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_mesh_bearer_platform_api_t::close) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl)

Close Bearer Platform middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ setScanResponseData

fsp_err_t(* rm_mesh_bearer_platform_api_t::setScanResponseData) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_data, uint8_t len)

Set scan response data in connectable and scannable undirected advertising event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_data | Pointer to scan response data. |
| [in] | len | Data length. |

### ◆ connect

fsp_err_t(* rm_mesh_bearer_platform_api_t::connect) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint8_t *p_remote_address, uint8_t address_type, rm_mesh_bearer_platform_gatt_mode_t mode)

Request to create connection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_remote_address | Pointer to remote device address. |
| [in] | address_type | Address type. |
| [in] | mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

### ◆ discoverService

fsp_err_t(* rm_mesh_bearer_platform_api_t::discoverService) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_gatt_mode_t mode)

Start service discovery for Mesh GATT service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Connection handle to identify device. |
| [in] | mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

### ◆ configureNotification

fsp_err_t(* rm_mesh_bearer_platform_api_t::configureNotification) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle, rm_mesh_bearer_platform_state_t state, rm_mesh_bearer_platform_gatt_mode_t mode)

Configure GATT notification of Mesh GATT service.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Connection handle to identify device. |
| [in] | state | Notification configuration flag; enable if RM_MESH_BEARER_PLATFORM_STATE_ENABLE, or disable if RM_MESH_BEARER_PLATFORM_STATE_DISABLE. |
| [in] | mode | GATT interface mode, either RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION or RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY. |

### ◆ disconnect

fsp_err_t(* rm_mesh_bearer_platform_api_t::disconnect) (rm_mesh_bearer_platform_ctrl_t *const p_ctrl, uint16_t handle)

Terminate Connection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | handle | Connection handle to identify device. |

### ◆ rm_mesh_bearer_platform_instance_t

struct rm_mesh_bearer_platform_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_mesh_bearer_platform_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_mesh_bearer_platform_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_mesh_bearer_platform_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_mesh_bearer_platform_ctrl_t

| typedef void rm_mesh_bearer_platform_ctrl_t |
| --- |
| MESH BEARER PLATFORM control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_mesh_bearer_platform_device_address_type_t

| enum rm_mesh_bearer_platform_device_address_type_t | |
| --- | --- |
| Device address type | |
| Enumerator | |
| RM_MESH_BEARER_PLATFORM_DEVICE_ADDRESS_TYPE_PUBLIC | Public device address type |
| RM_MESH_BEARER_PLATFORM_DEVICE_ADDRESS_TYPE_RANDOM | Random device address type |

### ◆ rm_mesh_bearer_platform_state_t

| enum rm_mesh_bearer_platform_state_t | |
| --- | --- |
| State | |
| Enumerator | |
| RM_MESH_BEARER_PLATFORM_STATE_ENABLE | State enable |
| RM_MESH_BEARER_PLATFORM_STATE_DISABLE | state Disable |

### ◆ rm_mesh_bearer_platform_gatt_mode_t

| enum rm_mesh_bearer_platform_gatt_mode_t | |
| --- | --- |
| GATT mode | |
| Enumerator | |
| RM_MESH_BEARER_PLATFORM_GATT_MODE_PROVISION | Provision GATT mode |
| RM_MESH_BEARER_PLATFORM_GATT_MODE_PROXY | Proxy GATT mode |

## BLE Mesh Health Server Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Model Health Server functions.

# Summary

The BLE Mesh interface for the BLE Mesh Model Health Server (BLE MESH HEALTH SERVER) middleware provides BLE Mesh Model Health Server functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_model_health_callback_args_t |
| struct | rm_ble_mesh_health_server_self_test_t |
| struct | rm_ble_mesh_health_server_cfg_t |
| struct | rm_ble_mesh_health_server_api_t |
| struct | rm_ble_mesh_health_server_instance_t |

### Typedefs

| | |
|---|---|
| typedef void(* | rm_ble_mesh_health_server_self_test_function) (uint8_t test_id, uint16_t company_id) |
| typedef void | rm_ble_mesh_health_server_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_health_server_events_t |

### Data Structure Documentation

#### ◆ rm_ble_mesh_model_health_callback_args_t

| struct rm_ble_mesh_model_health_callback_args_t | | |
|---|---|---|
| Mesh model health server callback parameter definition | | |
| Data Fields | | |
| void * | p_context | |
| rm_ble_mesh_access_model_handle_t * | p_handle | Access model handle. |

| | | |
|---|---|---|
| rm_ble_mesh_health_server_events_t | event_type | Application events defined for Health Server Model. |
| uint8_t * | p_event_data | Event data. |
| uint16_t | event_data_length | Event data length. |

### ◆ rm_ble_mesh_health_server_self_test_t

| struct rm_ble_mesh_health_server_self_test_t | | |
|---|---|---|
| Health Server Self Test Function Structure. | | |
| Data Fields | | |
| uint8_t | test_id | Test ID |
| rm_ble_mesh_health_server_self_test_function | self_test_fn | Self Test Function |

### ◆ rm_ble_mesh_health_server_cfg_t

| struct rm_ble_mesh_health_server_cfg_t | |
|---|---|
| BLE mesh model health server configuration parameters. | |
| **Data Fields** | |
| rm_ble_mesh_access_instance_t const * | p_access_instance |
| | Access Layer instance structure. More... |
| | |
| rm_ble_mesh_health_server_self_test_t * | p_self_tests |
| | Health Server self test function structure. |
| | |
| uint16_t | company_id |
| | Company ID. |
| | |
| uint32_t | num_self_tests |
| | Number of self test. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |

| | |
|---|---|
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_access_instance

rm_ble_mesh_access_instance_t const* rm_ble_mesh_health_server_cfg_t::p_access_instance

Access Layer instance structure.

the parameters for initialization.

### ◆ rm_ble_mesh_health_server_api_t

struct rm_ble_mesh_health_server_api_t

Shared Interface definition for BLE MESH

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_health_server_ctrl_t *const p_ctrl, rm_ble_mesh_health_server_cfg_t const *const p_cfg) |
| fsp_err_t(* | close )(rm_ble_mesh_health_server_ctrl_t *const p_ctrl) |
| fsp_err_t(* | reportFault )(rm_ble_mesh_health_server_ctrl_t *const p_ctrl, const rm_ble_mesh_access_model_handle_t *const p_model_handle, uint8_t test_id, uint16_t company_id, uint8_t fault_code) |
| fsp_err_t(* | publishCurrentStatus )(rm_ble_mesh_health_server_ctrl_t *const p_ctrl, uint8_t *p_status, uint16_t length) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_health_server_api_t::open) (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, rm_ble_mesh_health_server_cfg_t const *const p_cfg)

API to open health server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_health_server_api_t::close) (rm_ble_mesh_health_server_ctrl_t *const p_ctrl)

API to close health server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ reportFault

fsp_err_t(* rm_ble_mesh_health_server_api_t::reportFault) (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, const rm_ble_mesh_access_model_handle_t *const p_model_handle, uint8_t test_id, uint16_t company_id, uint8_t fault_code)

API to report self-test fault.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_model_handle | Pointer to model handle identifying the health server model instance. |
| [in] | test_id | Identifier of the self-test. |
| [in] | company_id | Company identifier. |
| [in] | fault_code | Fault value indicating the error. |

### ◆ publishCurrentStatus

fsp_err_t(* rm_ble_mesh_health_server_api_t::publishCurrentStatus) (rm_ble_mesh_health_server_ctrl_t *const p_ctrl, uint8_t *p_status, uint16_t length)

API to publish current status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_status | Pointer to current status. |
| [in] | length | Status data length. |

### ◆ rm_ble_mesh_health_server_instance_t

struct rm_ble_mesh_health_server_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_health_server_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_health_server_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_health_server_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_health_server_self_test_function

typedef void(* rm_ble_mesh_health_server_self_test_function) (uint8_t test_id, uint16_t company_id)

Health Server Self Test Function.

### ◆ rm_ble_mesh_health_server_ctrl_t

typedef void rm_ble_mesh_health_server_ctrl_t

BLE MESH HEALTH SERVER control block. Allocate an instance specific control block to pass into the BLE mesh model health server API calls.

## Enumeration Type Documentation

### ◆ rm_ble_mesh_health_server_events_t

| enum rm_ble_mesh_health_server_events_t | |
|---|---|
| This section lists the Application Events defined for Health Server Model | |
| Enumerator | |
| RM_BLE_MESH_HEALTH_SERVER_SERVER_ATTENTION_START | Attention Start |
| RM_BLE_MESH_HEALTH_SERVER_SERVER_ATTENTION_RESTART | Attention Restart |
| RM_BLE_MESH_HEALTH_SERVER_SERVER_ATTENTION_STOP | Attention Stop |

## BLE Mesh Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh functions.

# Summary

The BLE Mesh interface for the BLE Mesh (BLE MESH) peripheral provides BLE Mesh functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_buffer_t |
| struct | rm_ble_mesh_access_state_transition_t |
| struct | rm_ble_mesh_cfg_t |
| struct | rm_ble_mesh_api_t |
| struct | rm_ble_mesh_instance_t |

### Macros

| | |
|---|---|
| #define | RM_BLE_MESH_DEVICE_UUID_SIZE |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_COMMON |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_BEARER |

| | |
|---|---|
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_NETWORK |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_LOWER_TRANS |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_UPPER_TRANS |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_ACCESS |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_PROVISION |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_GENERIC_MODEL |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_CONFIG_MODEL |
| #define | RM_BLE_MESH_ERROR_GROUP_MASK_HEALTH_MODEL |

### Typedefs

| | |
|---|---|
| typedef void * | rm_ble_mesh_timer_handle_t |
| typedef void | rm_ble_mesh_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_error_code_t |
| enum | rm_ble_mesh_feature_select_t |
| enum | rm_ble_mesh_feature_state_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_buffer_t

| struct rm_ble_mesh_buffer_t | | |
|---|---|---|
| Payload type | | |
| Data Fields | | |
| uint8_t * | payload | Payload Pointer |
| uint16_t | length | Payload Length |

### ◆ rm_ble_mesh_access_state_transition_t

| struct rm_ble_mesh_access_state_transition_t | |
|---|---|
| Access Status Transition Type | |
| **Data Fields** | |
| rm_ble_mesh_timer_handle_t | transition_timer_handle |

| | |
|---|---|
| uint8_t | transition_state |
| uint8_t | delay |
| uint8_t | transition_time |
| void(* | transition_start_callback )(void *) |
| void(* | transition_complete_callback )(void *) |
| void * | p_context |

## Field Documentation

### ◆ transition_timer_handle

rm_ble_mesh_timer_handle_t rm_ble_mesh_access_state_transition_t::transition_timer_handle

Transition Timer

### ◆ transition_state

uint8_t rm_ble_mesh_access_state_transition_t::transition_state

Transition State. Initial/delay/transition

### ◆ delay

uint8_t rm_ble_mesh_access_state_transition_t::delay

Delay

### ◆ transition_time

uint8_t rm_ble_mesh_access_state_transition_t::transition_time

Transition Time

### ◆ transition_start_callback

void(* rm_ble_mesh_access_state_transition_t::transition_start_callback) (void *)

Transition Start Callback

### ◆ transition_complete_callback

| void(* rm_ble_mesh_access_state_transition_t::transition_complete_callback) (void *) |
|---|
| Transition Complete Callback |

### ◆ p_context

| void* rm_ble_mesh_access_state_transition_t::p_context |
|---|
| Blob/Context |

### ◆ rm_ble_mesh_cfg_t

| struct rm_ble_mesh_cfg_t | | |
|---|---|---|
| BLE MESH configuration parameters. | | |
| Data Fields | | |
| uint32_t | channel | Select a channel corresponding to the channel number of the hardware.<br><br>the parameters for initialization. |
| uint32_t | network_interfaces_num | Number of Network Interfaces |
| uint32_t | provisioning_interfaces_num | Number of Provisioning Interfaces |
| uint32_t | network_cache_size | Network Cache Size |
| uint32_t | network_sequence_num_cache_size | Network Sequence Number Cache Size |
| uint32_t | maximum_subnets | Maximum number of subnets the device can store information about. |
| uint32_t | maximum_device_keys | Maximum number of device keys the device can store information about. |
| uint32_t | proxy_filter_list_size | Maximum number of addresses present in each proxy filter list. |
| uint32_t | maximum_lpn | Maximum number of LPNs. |
| uint32_t | reassembled_cache_size | The size of Reassembled Cache. |
| uint32_t | maximum_ltrn_sar_context | Number of Segmentation and Reassembly contexts. |
| uint32_t | maximum_friend_message_queue | Maximum number of messages that the friend is capable to queue for a single Low Power Node. |

| | | |
|---|---|---|
| uint32_t | maximum_friend_subscription_list | Maximum number of subscription addresses that the friend is capable to store for a single Low Power Node. |
| uint32_t | maximum_access_element_num | Maximum number of elements. |
| uint32_t | maximum_access_model_num | Maximum number of models. |
| uint32_t | maximum_application | Maximum number of Applications (keys) the device can store information about. |
| uint32_t | maximum_virtual_address | Maximum number of Virtual Addresses the device can store information about. |
| uint32_t | maximum_non_virtual_address | Maximum number of Non-Virtual Addresses the device can store information about. |
| uint32_t | net_sequence_number_block_size | The distance between the network sequence numbers, for every persistent storage write. If the device is powered cycled, it will resume transmission using the sequence number from start of next block. |
| uint32_t | net_tx_count | Network Transmit Count for network packets |
| uint32_t | net_tx_interval_steps | Network Interval Steps for network packets |
| uint32_t | net_relay_tx_count | Network Transmit Count for relayed packets |
| uint32_t | net_relay_tx_interval_steps | Network Interval Steps for relayed packets |
| uint32_t | config_server_snb_timeout | Secure Network Beacon Interval |
| uint32_t | proxy_subnet_netid_adv_timeout | Poxy ADV Network ID timeout for each Subnet in milliseconds. |
| uint32_t | proxy_subnet_nodeid_adv_timeout | Poxy ADV Node Identity timeout for each Subnet in milliseconds. |
| uint32_t | proxy_nodeid_adv_timeout | Poxy ADV Node Identity overall time period in milliseconds. |
| uint32_t | frnd_poll_retry_count | Friend Poll Retry Count - default value |
| uint32_t | ltrn_rtx_timeout | Lower Transport Segment |

| | | |
|---|---|---|
| | | Transmission Timeout in milliseconds |
| uint32_t | ltrn_rtx_count | Lower Transport Segment Transmission Count - default value |
| uint32_t | ltrn_ack_timeout | Lower Transport Acknowledgment Timeout in milliseconds |
| uint32_t | ltrn_incomplete_timeout | Lower Transport Incomplete Timeout in milliseconds |
| uint32_t | frnd_receive_window | Friendship Receive Window - default value |
| uint32_t | lpn_clear_retry_timeout_initial | Friend Clear Confirmation Timeout in milliseconds |
| uint32_t | lpn_clear_retry_count | LPN Friend Clear Retry Count |
| uint32_t | trn_frndreq_retry_timeout | Friendship Retry Timeout in milliseconds |
| uint32_t | unprov_device_beacon_timeout | Unprovisioned Device Beacon Interleaved Beacon Timeout |
| uint32_t | net_tx_queue_size | Maximum number of messages that can be queued in Network layer for Transmission. |
| uint32_t | max_num_transition_timers | Maximum number of Transition Timers |
| uint32_t | max_num_periodic_step_timers | Maximum number of Periodic Step Timers |
| uint32_t | maximum_health_server_num | Maximum number of Health Server Instances. |
| uint32_t | maximum_light_lc_server_num | Maximum number of Light Lightness Controller Server Instances. |
| uint32_t | replay_cache_size | The size of the Replay Protection cache. |
| uint32_t | default_company_id | Company ID |
| uint32_t | default_product_id | Product ID |
| uint32_t | default_vendor_id | Vendor ID |
| void * | p_context | Placeholder for user data. |
| void const * | p_extend | Placeholder for user extension. |

### ◆ rm_ble_mesh_api_t

| struct rm_ble_mesh_api_t |
|---|
| BLE MESH functions implemented at the HAL layer will follow this API. |

| Data Fields | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | startTransitionTimer )(rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_access_state_transition_t const *const p_transition, uint16_t *const p_transition_time_handle) |
| | |
| fsp_err_t(* | stopTransitionTimer )(rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle) |
| | |
| fsp_err_t(* | getRemainingTransitionTime )(rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint8_t *const p_remaining_transition_time) |
| | |
| fsp_err_t(* | getRemainingTransitionTimeWithOffset )(rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint32_t offset_in_ms, uint8_t *const p_remaining_transition_time) |
| | |
| fsp_err_t(* | convertTransitionTimeFromMs )(rm_ble_mesh_ctrl_t *const p_ctrl, uint32_t transition_time_in_ms, uint8_t *const p_transition_time) |
| | |
| fsp_err_t(* | convertTransitionTimeToMs )(rm_ble_mesh_ctrl_t *const p_ctrl, uint8_t transition_time, uint32_t *const p_transition_time_in_ms) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_api_t::open) (rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_cfg_t const *const p_cfg)

Open BLE mesh middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_api_t::close) (rm_ble_mesh_ctrl_t *const p_ctrl)

Close BLE mesh middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ startTransitionTimer

fsp_err_t(* rm_ble_mesh_api_t::startTransitionTimer) (rm_ble_mesh_ctrl_t *const p_ctrl, rm_ble_mesh_access_state_transition_t const *const p_transition, uint16_t *const p_transition_time_handle)

To start transition timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_transition | Pointer to state transition data structure, which includes the timeout, transition start and complete callback etc. |
| [out] | p_transition_time_handle | Pointer to transition time handle, which can be used to stop the transition timer if required. |

### ◆ stopTransitionTimer

fsp_err_t(* rm_ble_mesh_api_t::stopTransitionTimer) (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle)

To stop transition timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | transition_time_handle | Transition time handle, returned by the Start Transition Timer interface. |

### ◆ getRemainingTransitionTime

fsp_err_t(* rm_ble_mesh_api_t::getRemainingTransitionTime) (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint8_t *const p_remaining_transition_time)

To get remaining Transition Time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | transition_time_handle | Transition time handle, returned by the Start Transition Timer interface. |
| [out] | p_remaining_transition_time | Pointer to remaining transition time. |

### ◆ getRemainingTransitionTimeWithOffset

fsp_err_t(* rm_ble_mesh_api_t::getRemainingTransitionTimeWithOffset) (rm_ble_mesh_ctrl_t *const p_ctrl, uint16_t transition_time_handle, uint32_t offset_in_ms, uint8_t *const p_remaining_transition_time)

To get remaining Transition Time, with offset.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | transition_time_handle | Transition time handle, returned by the Start Transition Timer interface. |
| [in] | offset_in_ms | Offset in millisecond. |
| [out] | p_remaining_transition_time | Pointer to remaining transition time. |

### ◆ convertTransitionTimeFromMs

fsp_err_t(* rm_ble_mesh_api_t::convertTransitionTimeFromMs) (rm_ble_mesh_ctrl_t *const p_ctrl, uint32_t transition_time_in_ms, uint8_t *const p_transition_time)

To convert transition time from millisecond.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | transition_time_in_ms | Transition time in millisecond. |
| [out] | p_transition_time | Pointer to converted value in Generic Default Transition Time state format. |

### ◆ convertTransitionTimeToMs

fsp_err_t(* rm_ble_mesh_api_t::convertTransitionTimeToMs) (rm_ble_mesh_ctrl_t *const p_ctrl, uint8_t transition_time, uint32_t *const p_transition_time_in_ms)

To convert transition time to millisecond.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | transition_time | Transition time in Generic Default Transition Time state format. |
| [out] | p_transition_time_in_ms | Pointer to converted value of transition time in millisecond. |

### ◆ rm_ble_mesh_instance_t

struct rm_ble_mesh_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ RM_BLE_MESH_DEVICE_UUID_SIZE

#define RM_BLE_MESH_DEVICE_UUID_SIZE

Device UUID Size

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK

#define RM_BLE_MESH_ERROR_GROUP_MASK

Error group mask for BLE MESH Module

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_COMMON

| #define RM_BLE_MESH_ERROR_GROUP_MASK_COMMON |
| --- |
| Error group mask for Common |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_BEARER

| #define RM_BLE_MESH_ERROR_GROUP_MASK_BEARER |
| --- |
| Error group mask for Bearer |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_NETWORK

| #define RM_BLE_MESH_ERROR_GROUP_MASK_NETWORK |
| --- |
| Error group mask for Network |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_LOWER_TRANS

| #define RM_BLE_MESH_ERROR_GROUP_MASK_LOWER_TRANS |
| --- |
| Error group mask for Lower Transport |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_UPPER_TRANS

| #define RM_BLE_MESH_ERROR_GROUP_MASK_UPPER_TRANS |
| --- |
| Error group mask for Upper Transport |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_ACCESS

| #define RM_BLE_MESH_ERROR_GROUP_MASK_ACCESS |
| --- |
| Error group mask for Access |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_PROVISION

| #define RM_BLE_MESH_ERROR_GROUP_MASK_PROVISION |
| --- |
| Error group mask for Provisioning |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_GENERIC_MODEL

| #define RM_BLE_MESH_ERROR_GROUP_MASK_GENERIC_MODEL |
| --- |
| Error group mask for Generic Models |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_CONFIG_MODEL

| #define RM_BLE_MESH_ERROR_GROUP_MASK_CONFIG_MODEL |
| --- |
| Error group mask for Config Model |

### ◆ RM_BLE_MESH_ERROR_GROUP_MASK_HEALTH_MODEL

| #define RM_BLE_MESH_ERROR_GROUP_MASK_HEALTH_MODEL |
| --- |
| Error group mask for Health Model |

## Typedef Documentation

### ◆ rm_ble_mesh_timer_handle_t

| typedef void* rm_ble_mesh_timer_handle_t |
| --- |
| Timer handle |

### ◆ rm_ble_mesh_ctrl_t

| typedef void rm_ble_mesh_ctrl_t |
| --- |
| BLE MESH control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_error_code_t

| enum rm_ble_mesh_error_code_t | |
|---|---|
| BLE MESH error code. | |
| Enumerator | |
| RM_BLE_MESH_ERROR_CODE_SUCCESS | Success |
| RM_BLE_MESH_ERROR_CODE_INVALID_ADDRESS | Invalid Address |
| RM_BLE_MESH_ERROR_CODE_INVALID_MODEL | Invalid Model |
| RM_BLE_MESH_ERROR_CODE_INVALID_APPKEY_INDEX | Invalid AppKey Index |
| RM_BLE_MESH_ERROR_CODE_INVALID_NETKEY_INDEX | Invalid NetKey Index |
| RM_BLE_MESH_ERROR_CODE_INSUFFICIENT_RESOURCES | Insufficient Resources |
| RM_BLE_MESH_ERROR_CODE_KEY_INDEX_ALREADY_STORED | Key Index Already Stored |
| RM_BLE_MESH_ERROR_CODE_INVALID_PUBLISH_PARAMETER | Invalid Publish Parameters |
| RM_BLE_MESH_ERROR_CODE_NOT_A_SUBSCRIBE_MODEL | Not a Subscribe Model |
| RM_BLE_MESH_ERROR_CODE_STORAGE_FAILURE | Storage Failure |
| RM_BLE_MESH_ERROR_CODE_FEATURE_NOT_SUPPORTED | Feature Not Supported |
| RM_BLE_MESH_ERROR_CODE_CANNOT_UPDATE | Cannot Update |
| RM_BLE_MESH_ERROR_CODE_CANNOT_REMOVE | Cannot Remove |
| RM_BLE_MESH_ERROR_CODE_CANNOT_BIND | Cannot Bind |
| RM_BLE_MESH_ERROR_CODE_TEMP_UNABLE_TO_CHANGE_STATE | Temporarily Unable to Change State |
| RM_BLE_MESH_ERROR_CODE_CANNOT_SET | Cannot Set |
| RM_BLE_MESH_ERROR_CODE_UNSPECIFIED_ERROR | Unspecified Error |
| RM_BLE_MESH_ERROR_CODE_INVALID_BINDING | Invalid Binding |
| RM_BLE_MESH_ERROR_CODE_BEARER_MUTEX_INIT_FAILED | Bearer Error Code for MUTEX Initialization Failure |

| | |
|---|---|
| RM_BLE_MESH_ERROR_CODE_BEARER_COND_INIT_FAILED | Bearer Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_BEARER_MUTEX_LOCK_FAILED | Bearer Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_BEARER_MUTEX_UNLOCK_FAILED | Bearer Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_BEARER_MEMORY_ALLOCATION_FAILED | Bearer Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_BEARER_INVALID_PARAMETER_VALUE | Bearer Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_BEARER_PARAMETER_OUTSIDE_RANGE | Bearer Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_BEARER_NULL_PARAMETER_NOT_ALLOWED | Bearer Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_BEARER_INTERFACE_NOT_READY | Bearer Error Code for Interface Not Read |
| RM_BLE_MESH_ERROR_CODE_BEARER_API_NOT_SUPPORTED | Bearer Error Code for API Not Supported |
| RM_BLE_MESH_ERROR_CODE_NETWORK_MUTEX_INIT_FAILED | Network Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_NETWORK_COND_INIT_FAILED | Network Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_NETWORK_MUTEX_LOCK_FAILED | Network Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_NETWORK_MUTEX_UNLOCK_FAILED | Network Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_NETWORK_MEMORY_ALLOCATION_FAILED | Network Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_NETWORK_INVALID_PARAMETER_VALUE | Network Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_NETWORK_PARAMETER_OUTSIDE_RANGE | Network Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_NETWORK_NULL_PARAMETER_NOT_ALLOWED | Network Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_NETWORK_TX_QUEUE_FULL | Network Error Code for Transmit Queue Full |
| RM_BLE_MESH_ERROR_CODE_NETWORK_TX_QU | |

| | |
|---|---|
| EUE_EMPTY | Network Error Code for Transmit Queue Empty |
| RM_BLE_MESH_ERROR_CODE_NETWORK_INVALID_RX_PACKET_FORMAT | Network Error Code returned by Network Callback, indicating if it detected an invalid packet format or if the packet to be further processed, by the network layer like to be relayed or proxied etc. |
| RM_BLE_MESH_ERROR_CODE_NETWORK_RX_LOCAL_SRC_ADDR_PACKET | Network Error Code for reception of locally sourced packet |
| RM_BLE_MESH_ERROR_CODE_NETWORK_POST_PROCESS_RX_PACKET | Network Error Code returned by Network Callback, indicating if the received packet needs further processing by the network layer like to be relayed or proxied etc. after the control being returned by the callback. |
| RM_BLE_MESH_ERROR_CODE_NETWORK_RX_ALREADY_RELAYED_PACKET | Network Error Code for Already Relayed Packet |
| RM_BLE_MESH_ERROR_CODE_NETWORK_RX_LPN_SRC_ADDR_TO_RELAY_PACKET | Network Error Code returned by Network Callback, indicating if the received packet is for a known LPN with an ongoing Friendship which needs further processing by the network layer like to be relayed or proxied etc. after the control being returned by the callback. |
| RM_BLE_MESH_ERROR_CODE_NETWORK_CRYPTO_UNLIKELY_ERR | Network Error code to tag all errors returned by the Crypto Interface to the Network layer. |
| RM_BLE_MESH_ERROR_CODE_NETWORK_API_NOT_SUPPORTED | Network Error Code for API Not Supported |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_MUTEX_INIT_FAILED | Lower Transport Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_COND_INIT_FAILED | Lower Transport Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_MUTEX_LOCK_FAILED | Lower Transport Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_MUTEX_UNLOCK_FAILED | Lower Transport Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_MEMORY_ALLOCATION_FAILED | Lower Transport Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_INVALID_PARAMETER_VALUE | Lower Transport Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_PA | Lower Transport Error Code for Parameter |

| | |
|---|---|
| RAMETER_OUTSIDE_RANGE | Outside Range |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_NULL_PARAMETER_NOT_ALLOWED | Lower Transport Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_LOWER_TRANS_SAR_CTX_ALLOCATION_FAILED | Lower Transport Error Code for SAR Context Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_MUTEX_INIT_FAILED | Upper Transport Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_COND_INIT_FAILED | Upper Transport Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_MUTEX_LOCK_FAILED | Upper Transport Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_MUTEX_UNLOCK_FAILED | Upper Transport Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_MEMORY_ALLOCATION_FAILED | Upper Transport Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_INVALID_PARAMETER_VALUE | Upper Transport Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_PARAMETER_OUTSIDE_RANGE | Upper Transport Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_NULL_PARAMETER_NOT_ALLOWED | Upper Transport Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_QUEUE_FULL | Upper Transport Error Code for Queue Full |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_QUEUE_EMPTY | Upper Transport Error Code for Queue Empty |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_INCOMPLETE_PACKET_RECEIVED | Upper Transport Error Code for Incomplete Packet Reception |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_INVALID_FRNDSHIP_STATE | Upper Transport Error Code for Invalid Friendship State |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_FRNDSHIP_CLEARED_ON_TIMEOUT | Upper Transport Error Code for Friendship Cleared on Timeout |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_CRYPTO_UNLIKELY_ERR | Upper Transport Error code to tag all errors returned by the Crypto Interface to the Transport layer. |
| RM_BLE_MESH_ERROR_CODE_UPPER_TRANS_API | Upper Transport Error Code for API Not |

| | |
|---|---|
| _NOT_SUPPORTED | Supported |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MUTEX_INIT_FAILED | Access Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_ACCESS_COND_INIT_FAILED | Access Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MUTEX_LOCK_FAILED | Access Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MUTEX_UNLOCK_FAILED | Access Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MEMORY_ALLOCATION_FAILED | Access Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_ACCESS_INVALID_PARAMETER_VALUE | Access Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_ACCESS_PARAMETER_OUTSIDE_RANGE | Access Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_ACCESS_NULL_PARAMETER_NOT_ALLOWED | Access Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_ACCESS_NO_RESOURCE | Access Error Code for No Resources |
| RM_BLE_MESH_ERROR_CODE_ACCESS_NO_MATCH | Access Error Code for No Match |
| RM_BLE_MESH_ERROR_CODE_ACCESS_INVALID_HANDLE | Access Error Code for Invalid Handle |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MODEL_ALREADY_REGISTERED | Access Error Code for Model Already Registered |
| RM_BLE_MESH_ERROR_CODE_ACCESS_INVALID_SRC_ADDR | Access Error Code for Invalid Source Address |
| RM_BLE_MESH_ERROR_CODE_ACCESS_DEV_KEY_TABLE_FULL | Access Error Code for Device Key Table Full |
| RM_BLE_MESH_ERROR_CODE_ACCESS_MASTER_NID_ON_LPN | Access Error Code when detecting Packets with Master Network Credentials from a known address with ongoing Friendship |
| RM_BLE_MESH_ERROR_CODE_ACCESS_INVALID_PUBLICATION_STATE | Access Error Code for Invalid Publication State |
| RM_BLE_MESH_ERROR_CODE_ACCESS_INVALID_PUBLICATION_TTL | Access Error Code for Invalid Publication TTL |
| RM_BLE_MESH_ERROR_CODE_ACCESS_IV_VAL_NOT_PERMITTED | Access Error Code for not permitted IV Index Value |

| | |
|---|---|
| RM_BLE_MESH_ERROR_CODE_ACCESS_IV_UPDATE_TOO_SOON | Access Error Code for IV Update occurring too soon within the stipulated/specified Time Limit |
| RM_BLE_MESH_ERROR_CODE_ACCESS_IV_INCORRECT_STATE | Access Error Code for IV Update in Incorrect State |
| RM_BLE_MESH_ERROR_CODE_ACCESS_IV_UPDATE_DEFERRED_IN_BUSY | Access Error Code for IV Update deferred due to currently Busy State |
| RM_BLE_MESH_ERROR_CODE_ACCESS_API_NOT_SUPPORTED | Access Error Code for API Not Supported |
| RM_BLE_MESH_ERROR_CODE_PROVISION_MUTEX_INIT_FAILED | Provisioning Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_COND_INIT_FAILED | Provisioning Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_MUTEX_LOCK_FAILED | Provisioning Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_MUTEX_UNLOCK_FAILED | Provisioning Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_MEMORY_ALLOCATION_FAILED | Provisioning Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_INVALID_STATE | Provisioning Error Code for Invalid State |
| RM_BLE_MESH_ERROR_CODE_PROVISION_INVALID_PARAMETER | Bearer Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_PROVISION_CONTEXT_ALLOC_FAILED | Provisioning Error Code for Context Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_CONTEXT_ASSERT_FAILED | Provisioning Error Code for Context Assertion Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_CONTEXT_LINK_OPEN | Provisioning Error Code for Context Link Open |
| RM_BLE_MESH_ERROR_CODE_PROVISION_BEARER_ASSERT_FAILED | Provisioning Error Code for Bearer Assertion Failure |
| RM_BLE_MESH_ERROR_CODE_PROVISION_PROCEDURE_TIMEOUT | Provisioning Error Code for Procedure Timeout |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_MUTEX_INIT_FAILED | Generic Model Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_COND_INIT_FAILED | Generic Model Error Code for Conditional Variable Initialization Failure |

| | |
|---|---|
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_MUTEX_LOCK_FAILED | Generic Model Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_MUTEX_UNLOCK_FAILED | Generic Model Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_MEMORY_ALLOCATION_FAILED | Generic Model Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_INVALID_STATE | Generic Model Error Code for Invalid State |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_INVALID_PARAMETER | Generic Model Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_PARAMETER_OUTSIDE_RANGE | Generic Model Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_NULL_PARAMETER_NOT_ALLOWED | Generic Model Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_INVALID_MODEL_HANDLE | Generic Model Error Code for Invalid Model Handle |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_INVALID_ELEMENT_HANDLE | Generic Model Error Code for Invalid Element Handle |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_NOT_PRIMARY_ELEMENT | Generic Model Error Code when Operation is detected on a Node element which is not Primary Element |
| RM_BLE_MESH_ERROR_CODE_GENERIC_MODEL_API_NOT_SUPPORTED | Generic Model Error Code for API Not Supported |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_MUTEX_INIT_FAILED | Config Model Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_COND_INIT_FAILED | Config Model Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_MUTEX_LOCK_FAILED | Config Model Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_MUTEX_UNLOCK_FAILED | Config Model Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_MEMORY_ALLOCATION_FAILED | Config Model Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_INVALID_STATE | Config Model Error Code for Invalid State |

| | |
|---|---|
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_INVALID_PARAMETER | Config Model Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_PARAMETER_OUTSIDE_RANGE | Config Model Error Code for Parameter Outside Range |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_NULL_PARAMETER_NOT_ALLOWED | Config Model Error Code for NULL Parameter Not Allowed |
| RM_BLE_MESH_ERROR_CODE_CONFIG_MODEL_API_NOT_SUPPORTED | Config Model Error Code for API Not Supported |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_MUTEX_INIT_FAILED | Health Model Error Code for MUTEX Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_COND_INIT_FAILED | Health Model Error Code for Conditional Variable Initialization Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_MUTEX_LOCK_FAILED | Health Model Error Code for MUTEX Lock Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_MUTEX_UNLOCK_FAILED | Health Model Error Code for MUTEX Unlock Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_MEMORY_ALLOCATION_FAILED | Health Model Error Code for Memory Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_INVALID_STATE | Health Model Error Code for Invalid State |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_INVALID_PARAMETER | Health Model Error Code for Invalid Parameter Value |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_CONTEXT_ALLOC_FAILED | Health Model Error Code for Context Allocation Failure |
| RM_BLE_MESH_ERROR_CODE_HEALTH_MODEL_CONTEXT_ASSERT_FAILED | Health Model Error Code for Context Assertion Failure |
| RM_BLE_MESH_ERROR_CODE_FAILURE | A failure that does not match any other error code |

### ◆ rm_ble_mesh_feature_select_t

| enum rm_ble_mesh_feature_select_t | |
|---|---|
| Node Feature | |
| Enumerator | |
| RM_BLE_MESH_FEATURE_SELECT_RELAY | Relay Feature |
| RM_BLE_MESH_FEATURE_SELECT_PROXY | Proxy Feature |
| RM_BLE_MESH_FEATURE_SELECT_FRIEND | Friend Feature |
| RM_BLE_MESH_FEATURE_SELECT_LPN | Low Power Feature |

### ◆ rm_ble_mesh_feature_state_t

| enum rm_ble_mesh_feature_state_t | |
|---|---|
| Enumerator | |
| RM_BLE_MESH_FEATURE_STATE_DISABLE | BLE Mesh featre status - Disable |
| RM_BLE_MESH_FEATURE_STATE_ENABLE | BLE Mesh featre status - Enable |

## BLE Mesh Lower Trans Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Lower Trans functions.

# Summary

The BLE Mesh Lower Trans middleware provides a lower-level transfer interface for BLE Mesh services.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_lower_trans_transmit_setting_t |
| struct | rm_ble_mesh_lower_trans_callback_args_t |
| struct | rm_ble_mesh_lower_trans_cfg_t |

| | |
|---|---|
| struct | rm_ble_mesh_lower_trans_api_t |
| struct | rm_ble_mesh_lower_trans_instance_t |

### Typedefs

| | |
|---|---|
| typedef uint8_t | rm_ble_mesh_lower_trans_lpn_handle_t |
| typedef void | rm_ble_mesh_lower_trans_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_lower_trans_message_type_t |
| enum | rm_ble_mesh_lower_trans_reliable_t |
| enum | rm_ble_mesh_lower_trans_event_t |
| enum | rm_ble_mesh_lower_trans_notification_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_lower_trans_transmit_setting_t

| struct rm_ble_mesh_lower_trans_transmit_setting_t | | |
|---|---|---|
| transport PDUs setting to peer device. | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | src_addr | Source address. |
| rm_ble_mesh_network_address_t | dst_addr | Destination address. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Handle identifying the subnet. |
| rm_ble_mesh_lower_trans_message_type_t | msg_type | Transport message type. |
| uint8_t | ttl | Time to Live. |
| uint8_t | akf | Application key flag. |
| uint8_t | aid | Application key identifier. |
| uint8_t | seq_num | Sequence number. |

### ◆ rm_ble_mesh_lower_trans_callback_args_t

| struct rm_ble_mesh_lower_trans_callback_args_t |
|---|
| BLE Mesh lower trans callback parameter definition |
| Data Fields |

| | | |
|---|---|---|
| rm_ble_mesh_lower_trans_event_t | event | Event code. |
| rm_ble_mesh_network_header_t * | p_header | Network header. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Subnet handle. |
| uint8_t | trans_mic_size | TransMIC size. |
| rm_ble_mesh_lower_trans_lpn_handle_t | lpn_handle | LPM Size. |
| uint8_t | packet_type | packet type |
| rm_ble_mesh_buffer_t | event_data | Event data. |
| void * | p_context | Context provided to user during callback. |

### ◆ rm_ble_mesh_lower_trans_cfg_t

| struct rm_ble_mesh_lower_trans_cfg_t | |
|---|---|
| BLE MESH configuration parameters. | |
| **Data Fields** | |
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| rm_ble_mesh_network_instance_t const * | p_mesh_network_instance |
| | Instance structure of BLE Mesh network. |
| | |
| void(* | p_callback )(rm_ble_mesh_lower_trans_callback_args_t *p_args) |
| | Callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Placeholder for extension data. |

## Field Documentation

### ◆ channel

uint32_t rm_ble_mesh_lower_trans_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

### ◆ rm_ble_mesh_lower_trans_api_t

struct rm_ble_mesh_lower_trans_api_t

BLE MESH functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_cfg_t const *const p_cfg) |
| fsp_err_t(* | close )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |
| fsp_err_t(* | sendPdu )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_transmit_setting_t const *const p_transmit_setting, rm_ble_mesh_buffer_t const *const p_buffer, rm_ble_mesh_lower_trans_reliable_t reliable) |
| fsp_err_t(* | clearSarContexts )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |
| fsp_err_t(* | clearSubnetSarContexts )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| fsp_err_t(* | reinitReplayCache )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |
| fsp_err_t(* | triggerPendingTransmits )(rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::open) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_cfg_t const *const p_cfg)

Register Interface with Lower Transport Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::close) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl)

Unregister Interface with Lower Transport Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ sendPdu

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::sendPdu) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_lower_trans_transmit_setting_t const *const p_transmit_setting, rm_ble_mesh_buffer_t const *const p_buffer, rm_ble_mesh_lower_trans_reliable_t reliable)

API to send transport PDUs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_transmit_setting | Pointer to transmit setting structure. |
| [in] | p_buffer | Pointer to payload and payload length structure. |
| [in] | reliable | If requires lower transport ACK, set reliable as RM_BLE_MESH_LOWER_TRANS_RELIABLE_ENABLE. |

### ◆ clearSarContexts

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::clearSarContexts) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl)

To clear all segmentation and reassembly contexts.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ clearSubnetSarContexts

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::clearSubnetSarContexts) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle)

To clear all segmentation and reassembly contexts for a given subnet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Subnet Handle whose respective SAR Contexts are to be cleared. |

### ◆ reinitReplayCache

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::reinitReplayCache) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl)

To reinitialize all Lower Transport replay cache entries.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ triggerPendingTransmits

fsp_err_t(* rm_ble_mesh_lower_trans_api_t::triggerPendingTransmits) (rm_ble_mesh_lower_trans_ctrl_t *const p_ctrl)

To trigger any Lower Transport pending transmissions.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_ble_mesh_lower_trans_instance_t

struct rm_ble_mesh_lower_trans_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| rm_ble_mesh_lower_trans_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_lower_trans_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_lower_trans_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_lower_trans_lpn_handle_t

| typedef uint8_t rm_ble_mesh_lower_trans_lpn_handle_t |
|---|
| LPN handle |

### ◆ rm_ble_mesh_lower_trans_ctrl_t

| typedef void rm_ble_mesh_lower_trans_ctrl_t |
|---|
| BLE MESH control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_lower_trans_message_type_t

| enum rm_ble_mesh_lower_trans_message_type_t | |
|---|---|
| Transport Message Type | |
| Enumerator | |
| RM_BLE_MESH_LOWER_TRANS_MESSAGE_TYPE_ACCESS | Access Layer Packet |
| RM_BLE_MESH_LOWER_TRANS_MESSAGE_TYPE_CTRL | Transport Layer Control Packet |

### ◆ rm_ble_mesh_lower_trans_reliable_t

| enum rm_ble_mesh_lower_trans_reliable_t | |
|---|---|
| If requires lower transport ACK | |
| Enumerator | |
| RM_BLE_MESH_LOWER_TRANS_RELIABLE_DISABLE | ACK enable. |
| RM_BLE_MESH_LOWER_TRANS_RELIABLE_ENABLE | ACK disable. |

### ◆ rm_ble_mesh_lower_trans_event_t

| enum rm_ble_mesh_lower_trans_event_t | |
|---|---|
| Callback event | |
| Enumerator | |
| RM_BLE_MESH_LOWER_TRANS_EVENT_RX_COMPLETED | BLE Mesh Lower Trans event - RX completed. |
| RM_BLE_MESH_LOWER_TRANS_EVENT_RX_FROM_FRIEND | BLE Mesh Lower Trans event - RX from friend. |
| RM_BLE_MESH_LOWER_TRANS_EVENT_ADD_UPPER_TRANS_COMPLETED | BLE Mesh Lower Trans event - Add Upper Trans completed. |
| RM_BLE_MESH_LOWER_TRANS_EVENT_SEG_TX_CANCELED | BLE Mesh Lower Trans event - TX canceled. |
| RM_BLE_MESH_LOWER_TRANS_EVENT_SEG_TX_COMPLETED | BLE Mesh Lower Trans event - TX completed. |

### ◆ rm_ble_mesh_lower_trans_notification_t

| enum rm_ble_mesh_lower_trans_notification_t | |
|---|---|
| Whether to enable the event or not. | |
| Enumerator | |
| RM_BLE_MESH_LOWER_TRANS_NOTIFICATION_DISABLE | Callback disable. |
| RM_BLE_MESH_LOWER_TRANS_NOTIFICATION_ENABLE | Callback enable. |

## BLE Mesh Model Client Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Model Client functions.

# Summary

The BLE Mesh interface for the BLE Mesh Model Client (BLE MESH MODEL CLIENT) middleware provides BLE Mesh Model Client functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_model_client_callback_args_t |
| struct | rm_ble_mesh_model_client_cfg_t |
| struct | rm_ble_mesh_model_client_api_t |
| struct | rm_ble_mesh_model_client_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ble_mesh_model_client_ctrl_t |

### Data Structure Documentation

#### ◆ rm_ble_mesh_model_client_callback_args_t

| struct rm_ble_mesh_model_client_callback_args_t | | |
|---|---|---|
| Mesh model client callback parameter definition | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_req_msg_context_t * | p_msg_context | Context of message received for a specific model instance. |
| rm_ble_mesh_access_req_msg_raw_t * | p_msg_raw | Uninterpreted/raw received message for a specific model instance. |

#### ◆ rm_ble_mesh_model_client_cfg_t

| struct rm_ble_mesh_model_client_cfg_t | |
|---|---|
| BLE mesh model health client configuration parameters. | |
| **Data Fields** | |
| rm_ble_mesh_access_instance_t const * | p_access_instance |

| | |
|---|---|
| | Access Layer instance structure. More... |
| | |
| void(* | p_callback )(rm_ble_mesh_model_client_callback_args_t *p_args) |
| | Mesh model client callback. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ p_access_instance

| |
|---|
| rm_ble_mesh_access_instance_t const* rm_ble_mesh_model_client_cfg_t::p_access_instance |
| Access Layer instance structure.<br>the parameters for initialization. |

### ◆ rm_ble_mesh_model_client_api_t

| | |
|---|---|
| struct rm_ble_mesh_model_client_api_t | |
| Shared Interface definition for BLE MESH | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_model_client_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | getModelHandle )(rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle) |
| | |
| fsp_err_t(* | sendReliablePdu )(rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_model_client_api_t::open) (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_model_client_cfg_t const *const p_cfg)

API to open client model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_model_client_api_t::close) (rm_ble_mesh_model_client_ctrl_t *const p_ctrl)

API to close client model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ getModelHandle

fsp_err_t(* rm_ble_mesh_model_client_api_t::getModelHandle) (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, rm_ble_mesh_access_model_handle_t *const p_model_handle)

API to get Model client model handle.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_model_handle | Pointer to model handle to be filled/returned. |

### ◆ sendReliablePdu

| fsp_err_t(* rm_ble_mesh_model_client_api_t::sendReliablePdu) (rm_ble_mesh_model_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
|---|

API to send acknowledged commands.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | req_opcode | Request Opcode. |
| [in] | p_parameter | Pointer to Parameter associated with Request Opcode. |
| [in] | rsp_opcode | Response Opcode. |

### ◆ rm_ble_mesh_model_client_instance_t

| struct rm_ble_mesh_model_client_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_ble_mesh_model_client_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_model_client_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_model_client_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_model_client_ctrl_t

| typedef void rm_ble_mesh_model_client_ctrl_t |
|---|

BLE MESH MODEL CLIENT control block. Allocate an instance specific control block to pass into the BLE mesh model health client API calls.

## BLE Mesh Model Configuration Client Interface

Interfaces » Networking » BLE Mesh Network Interfaces

## Detailed Description

Interface for BLE Mesh Model Configuration Client functions.

# Summary

The BLE Mesh interface for the BLE Mesh Model Configuration Client (BLE MESH MODEL CONFIG CLIENT) middleware provides BLE Mesh Model Configuration Client functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_config_client_callback_args_t |
| struct | rm_ble_mesh_config_client_cfg_t |
| struct | rm_ble_mesh_config_client_api_t |
| struct | rm_ble_mesh_config_client_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ble_mesh_config_client_ctrl_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_config_client_callback_args_t

| struct rm_ble_mesh_config_client_callback_args_t | | |
|---|---|---|
| Mesh model client callback parameter definition | | |
| Data Fields | | |
| void * | p_context | |
| rm_ble_mesh_access_model_req_msg_context_t * | p_msg_context | Context of message received for a specific model instance. |
| rm_ble_mesh_access_req_msg_raw_t * | p_msg_raw | Uninterpreted/raw received message for a specific model instance. |

### ◆ rm_ble_mesh_config_client_cfg_t

| struct rm_ble_mesh_config_client_cfg_t | |
|---|---|
| BLE mesh model health client configuration parameters. | |
| **Data Fields** | |
| rm_ble_mesh_access_instance_t const * | p_access_instance |
| | |
| void(* | p_callback )(rm_ble_mesh_config_client_callback_args_t *p_args) |
| | Callback function. |

| | |
|---|---|
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_access_instance

rm_ble_mesh_access_instance_t const* rm_ble_mesh_config_client_cfg_t::p_access_instance

the parameters for initialization.

### ◆ rm_ble_mesh_config_client_api_t

struct rm_ble_mesh_config_client_api_t

Shared Interface definition for BLE MESH

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_config_client_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_config_client_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | setServer )(rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t server_addr, uint8_t *p_dev_key) |
| | |
| fsp_err_t(* | sendReliablePdu )(rm_ble_mesh_config_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_config_client_api_t::open) (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_config_client_cfg_t const *const p_cfg)

API to open configuration client model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_config_client_api_t::close) (rm_ble_mesh_config_client_ctrl_t *const p_ctrl)

API to close configuration client model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ setServer

fsp_err_t(* rm_ble_mesh_config_client_api_t::setServer) (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t server_addr, uint8_t *p_dev_key)

API to set configuration server.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | server_addr | Address of Configuration Server. |
| [in] | p_dev_key | Pointer to device Key of Configuration Server. |

### ◆ sendReliablePdu

fsp_err_t(* rm_ble_mesh_config_client_api_t::sendReliablePdu) (rm_ble_mesh_config_client_ctrl_t *const p_ctrl, uint32_t req_opcode, void const *const p_parameter, uint32_t rsp_opcode)

API to send acknowledged commands.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | req_opcode | Request Opcode. |
| [in] | p_parameter | Pointer to parameter associated with Request Opcode. |
| [in] | rsp_opcode | Response Opcode. |

### ◆ rm_ble_mesh_config_client_instance_t

struct rm_ble_mesh_config_client_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_config_client_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_config_client_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_config_client_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_config_client_ctrl_t

typedef void rm_ble_mesh_config_client_ctrl_t

BLE MESH CONFIG CLIENT control block. Allocate an instance specific control block to pass into the BLE mesh model health client API calls.

## BLE Mesh Model Server Interface

Interfaces » Networking » BLE Mesh Network Interfaces

## Detailed Description

Interface for BLE Mesh Model Server functions.

# Summary

The BLE Mesh interface for the BLE Mesh Model Server (BLE MESH MODEL SERVER) middleware provides BLE Mesh Model Server functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_model_server_generic_prop_srv_state_info_t |
| struct | rm_ble_mesh_model_server_generic_prop_srv_id_info_t |
| struct | rm_ble_mesh_model_server_generic_prop_srv_ids_info_t |
| struct | rm_ble_mesh_model_server_callback_args_t |
| struct | rm_ble_mesh_model_server_timeout_callback_args_t |
| struct | rm_ble_mesh_model_server_cfg_t |
| struct | rm_ble_mesh_model_server_api_t |
| struct | rm_ble_mesh_model_server_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ble_mesh_model_server_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_model_server_generic_server_property_t |
| enum | rm_ble_mesh_model_server_user_access_t |
| enum | rm_ble_mesh_model_server_device_property_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_model_server_generic_prop_srv_state_info_t

| struct rm_ble_mesh_model_server_generic_prop_srv_state_info_t |
|---|
| Generic Property is a state representing a device property of an element. The properties can be one of the following<br>• Manufacturer Properties<br>• Admin Properties<br>• User Properties |
| Data Fields |
| | | |

| | | |
|---|---|---|
| uint16_t | property_id | User Property ID field is a 2-octet Assigned Number value referencing a property |
| uint8_t | property_type | Property Type - Manufacturer/Admin/User |
| uint8_t | access | User Access field is an enumeration indicating whether the device property can be read or written as a Generic Admin/User Property |
| uint8_t * | property_value | User Property Value field is a conditional field |
| uint16_t | property_value_len | |

### ◆ rm_ble_mesh_model_server_generic_prop_srv_id_info_t

| struct rm_ble_mesh_model_server_generic_prop_srv_id_info_t | | |
|---|---|---|
| Generic Property ID a read-only state representing a device property that an element supports | | |
| Data Fields | | |
| uint16_t | property_id | Property ID field is a 2-octet Assigned Number value that references a device property |

### ◆ rm_ble_mesh_model_server_generic_prop_srv_ids_info_t

| struct rm_ble_mesh_model_server_generic_prop_srv_ids_info_t | | |
|---|---|---|
| Generic Property IDs a state representing a set of device properties that an element supports | | |
| Data Fields | | |
| uint16_t * | property_ids | Property IDs field is a set of 2-octet Assigned Number value that references a set of device properties |
| uint16_t | property_ids_count | Count of property_ids |

### ◆ rm_ble_mesh_model_server_callback_args_t

| struct rm_ble_mesh_model_server_callback_args_t | | |
|---|---|---|
| Mesh model server callback parameter definition | | |
| Data Fields | | |
| void * | p_context | |
| rm_ble_mesh_access_model_req_msg_context_t * | p_msg_context | Context of the message. |
| rm_ble_mesh_access_req_msg_raw_t * | p_msg_raw | Uninterpreted/raw received message for a specific model instance. |

| | | |
|---|---|---|
| rm_ble_mesh_access_model_req_msg_t * | p_req_type | Requested message type for a specific model instance. |
| rm_ble_mesh_access_model_state_parameter_t * | p_state_parameter | Model specific state parameters in a request or response message. |
| rm_ble_mesh_access_extended_parameter_t * | p_extended_parameter | Additional parameters in a Model specific request or response message. |

### ◆ rm_ble_mesh_model_server_timeout_callback_args_t

struct rm_ble_mesh_model_server_timeout_callback_args_t

Access Layer Model Publication Timeout Callback.

Access Layer calls the registered callback to indicate Publication Timeout for the associated model.

**Parameters**

| | |
|---|---|
| handle | Model Handle. |
| blob | Blob if any or NULL. |

Data Fields

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_handle_t * | p_handle | Access Model handle. |
| void * | p_blob | |

### ◆ rm_ble_mesh_model_server_cfg_t

struct rm_ble_mesh_model_server_cfg_t

BLE mesh model health server configuration parameters.

**Data Fields**

| | |
|---|---|
| rm_ble_mesh_access_instance_t const * | p_access_instance |
| | |
| void(* | p_callback )(rm_ble_mesh_model_server_callback_args_t *p_args) |
| | Mesh model server callback. |
| | |
| void(* | p_timeout_callback )(rm_ble_mesh_model_server_timeout_callback_args_t *p_args) |
| | Access Layer Model publication timeout callback. |
| | |

| | |
|---|---|
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_access_instance

rm_ble_mesh_access_instance_t const* rm_ble_mesh_model_server_cfg_t::p_access_instance

the parameters for initialization.

### ◆ rm_ble_mesh_model_server_api_t

struct rm_ble_mesh_model_server_api_t

Shared Interface definition for BLE MESH

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_model_server_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stateUpdate )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_model_server_api_t::open) (rm_ble_mesh_model_server_ctrl_t *const p_ctrl, rm_ble_mesh_model_server_cfg_t const *const p_cfg)

API to open server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_model_server_api_t::close) (rm_ble_mesh_model_server_ctrl_t *const p_ctrl)

API to close server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stateUpdate

fsp_err_t(* rm_ble_mesh_model_server_api_t::stateUpdate) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state)

API to send reply or to update state change.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_state | Pointer to model specific current/target state parameters. |

### ◆ rm_ble_mesh_model_server_instance_t

struct rm_ble_mesh_model_server_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_model_server_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_model_server_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_model_server_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_model_server_ctrl_t

typedef void rm_ble_mesh_model_server_ctrl_t

BLE MESH MODEL SERVER control block. Allocate an instance specific control block to pass into the BLE mesh model server API calls.

## Enumeration Type Documentation

### ◆ rm_ble_mesh_model_server_generic_server_property_t

| enum rm_ble_mesh_model_server_generic_server_property_t | |
|---|---|
| Enumerator | |
| RM_BLE_MESH_MODEL_SERVER_GENERIC_SERVER_PROPERTY_MANUFACTURER | Generic Manufacturer Properties |
| RM_BLE_MESH_MODEL_SERVER_GENERIC_SERVER_PROPERTY_ADMIN | Generic Admin Properties |
| RM_BLE_MESH_MODEL_SERVER_GENERIC_SERVER_PROPERTY_USER | Generic User Properties |

### ◆ rm_ble_mesh_model_server_user_access_t

| enum rm_ble_mesh_model_server_user_access_t | |
|---|---|
| User Access field values | |
| Enumerator | |
| RM_BLE_MESH_MODEL_SERVER_USER_ACCESS_PROHIBITED | User Access - Prohibited |
| RM_BLE_MESH_MODEL_SERVER_USER_ACCESS_READ | User Access - the device property can be read |
| RM_BLE_MESH_MODEL_SERVER_USER_ACCESS_WRITE | User Access - the device property can be written |
| RM_BLE_MESH_MODEL_SERVER_USER_ACCESS_READ_WRITE | User Access - the device property can be read and written |
| RM_BLE_MESH_MODEL_SERVER_USER_ACCESS_INVALID_PROPERTY_ID | User Access - the invalid device property id |

### ◆ rm_ble_mesh_model_server_device_property_t

| enum rm_ble_mesh_model_server_device_property_t | |
|---|---|
| Enumerator | |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_OCCUPANCY_DELAY | Device Property - Light Control Time Occupancy Delay |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_ON | Device Property - Light Control Time Fade On |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_RUN_ON | Device Property - Light Control Time Run On |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE | Device Property - Light Control Time Fade |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_PROLONG | Device Property - Light Control Time Prolong |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_STANDBY_AUTO | Device Property - Light Control Time Fade Standby Auto |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_TIME_FADE_STANDBY_MANUAL | Device Property - Light Control Time Fade Standby Manual |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_LIGHTNESS_ON | Device Property - Light Control Lightness On |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_LIGHTNESS_PROLONG | Device Property - Light Control Lightness Prolong |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_LIGHTNESS_STANDBY | Device Property - Light Control Lightness Standby |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_AMBIENT_LUXLEVEL_ON | Device Property - Light Control Ambient LuxLevel On |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_AMBIENT_LUXLEVEL_PROLONG | Device Property - Light Control Ambient LuxLevel Prolong |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_AMBIENT_LUXLEVEL_STANDBY | Device Property - Light Control Ambient LuxLevel Standby |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_REGULATOR_KIU | Device Property - Light Control Regulator Kiu |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_REGULATOR_KID | Device Property - Light Control Regulator Kid |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_REGULATOR_KPU | Device Property - Light Control Regulator Kpu |

| | |
|---|---|
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_REGULATOR_KPD | Device Property - Light Control Regulator Kpd |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_LIGHT_CONTROL_REGULATOR_ACCURACY | Device Property - Light Control Regulator Accuracy |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_MOTION_SENSED | Device Property - Motion Sensed |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_TIME_SINCE_MOTION_SENSED | Device Property - Time Since Motion Sensed |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_PEOPLE_COUNT | Device Property - People Count |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_PRESENCE_DETECTED | Device Property - Presence Detected |
| RM_BLE_MESH_MODEL_SERVER_DEVICE_PROPERTY_PRESENT_AMBIENT_LIGHT_LEVEL | Device Property - Present Ambient Light Level |

**BLE Mesh Network Interface**
Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Network functions.

# Summary

The BLE Mesh Network interface for the BLE Mesh Network (BLE MESH NETWORK) peripheral provides BLE Mesh Network functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_network_header_t |
| struct | rm_ble_mesh_network_seq_number_state_t |
| struct | rm_ble_mesh_network_route_info_t |
| struct | rm_ble_mesh_network_proxy_address_list_t |
| struct | rm_ble_mesh_network_callback_args_t |
| struct | rm_ble_mesh_network_cfg_t |
| struct | rm_ble_mesh_network_api_t |

| | |
|---|---|
| struct | rm_ble_mesh_network_instance_t |

### Macros

| | |
|---|---|
| #define | RM_BLE_MESH_NETWORK_UNASSIGNED_ADDRESS |
| #define | RM_BLE_MESH_NETWORK_PRIMARY_SUBNET |
| #define | RM_BLE_MESH_NETWORK_INVALID_SUBNET_HANDLE |
| #define | RM_BLE_MESH_NETWORK_INVALID_APPKEY_HANDLE |
| #define | RM_BLE_MESH_NETWORK_INVALID_NID |

### Typedefs

| | |
|---|---|
| typedef uint16_t | rm_ble_mesh_network_address_t |
| typedef uint16_t | rm_ble_mesh_network_subnet_handle_t |
| typedef uint16_t | rm_ble_mesh_network_appkey_handle_t |
| typedef uint16_t | rm_ble_mesh_network_proxy_address_t |
| typedef uint8_t | rm_ble_mesh_network_interface_handle_t |
| typedef void | rm_ble_mesh_network_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_network_old_packet_treatment_t |
| enum | rm_ble_mesh_network_rx_state_event_t |
| enum | rm_ble_mesh_network_tx_state_event_t |
| enum | rm_ble_mesh_network_address_type_t |
| enum | rm_ble_mesh_proxy_filter_type_t |
| enum | rm_ble_mesh_proxy_config_opcode_t |
| enum | rm_ble_mesh_network_gatt_proxy_state_t |
| enum | rm_ble_mesh_network_gatt_proxy_adv_mode_t |
| enum | rm_ble_mesh_network_event_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_network_header_t

| struct rm_ble_mesh_network_header_t | | |
|---|---|---|
| Network Header Type | | |
| Data Fields | | |
| uint8_t | ivi | Least significant bit of IV Index - 1 bit |
| uint8_t | nid | Value derived from the NetKey used to identify the Encrytion Key and Privacy Key used to secure this PDU - 7 bits |
| uint8_t | new_key | Indicates use of a new NetKey to which the network is being updated to. |
| uint8_t | ctl | Network Control - 1 bit |
| uint8_t | ttl | Time To Live - 7 bits |
| rm_ble_mesh_network_address_t | src_addr | 16 Bit Source Address |
| rm_ble_mesh_network_address_t | dst_addr | 16 Bit Destination Address |
| uint32_t | seq_num | 24 bit sequence number - currently filled only in reception path |

### ◆ rm_ble_mesh_network_seq_number_state_t

| struct rm_ble_mesh_network_seq_number_state_t | | |
|---|---|---|
| Current Sequence Number and Block State | | |
| Data Fields | | |
| uint32_t | seq_num | Current Sequence Number |
| uint32_t | block_seq_num_max | Block Sequence number - maximum available |

### ◆ rm_ble_mesh_network_route_info_t

| struct rm_ble_mesh_network_route_info_t | | |
|---|---|---|
| Network configuration information | | |
| Data Fields | | |
| rm_ble_mesh_network_interface_handle_t * | interface_handle | Pointer to list of address to be added/deleted |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Count of addresses present in the provided list |

### ◆ rm_ble_mesh_network_proxy_address_list_t

| struct rm_ble_mesh_network_proxy_address_list_t | | |
|---|---|---|
| Proxy Server's filter List | | |
| Data Fields | | |
| rm_ble_mesh_network_proxy_address_t * | address | Pointer to list of address to be added/deleted |
| uint16_t | count | Count of addresses present in the provided list |

### ◆ rm_ble_mesh_network_callback_args_t

| struct rm_ble_mesh_network_callback_args_t | | |
|---|---|---|
| BLE Mesh Network callback parameter definition | | |
| Data Fields | | |
| rm_ble_mesh_network_event_t | event | Event code. |
| rm_ble_mesh_network_header_t * | p_header | Network header type. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Subnet handle. |
| rm_ble_mesh_network_interface_handle_t * | p_network_interface | Network interface handle. |
| rm_ble_mesh_buffer_t | event_data | Event data. |
| void * | p_context | Context provided to user during callback. |

### ◆ rm_ble_mesh_network_cfg_t

| struct rm_ble_mesh_network_cfg_t | |
|---|---|
| BLE MESH NETWORK configuration parameters. | |
| **Data Fields** | |
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| rm_ble_mesh_bearer_instance_t const * | p_mesh_bearer_instance |
| | Instance structure of BLE Mesh Bearer. |
| | |
| rm_ble_mesh_provision_instance_t const * | p_mesh_provision_instance |
| | |

| | |
|---|---|
| | Instance structure of BLE Mesh Provision. |
| | |
| void(* | p_callback )(rm_ble_mesh_network_callback_args_t *p_args) |
| | Callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ channel

uint32_t rm_ble_mesh_network_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

### ◆ rm_ble_mesh_network_api_t

struct rm_ble_mesh_network_api_t

BLE MESH NETWORK functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_network_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | broadcastSecureBeacon )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle) |
| | |
| fsp_err_t(* | sendPduOnInterface )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_network_header_t const *const p_header, rm_ble_mesh_buffer_t const *const p_buffer) |

| | |
|---|---|
| fsp_err_t(* | getAddressType )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_network_address_type_t *const p_type) |
| fsp_err_t(* | fetchProxyState )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_gatt_proxy_state_t *const p_proxy_state) |
| fsp_err_t(* | setProxyFilter )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_filter_type_t type) |
| fsp_err_t(* | configProxyFilter )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_config_opcode_t opcode, rm_ble_mesh_network_proxy_address_list_t *const p_addr_list) |
| fsp_err_t(* | startProxyServerAdv )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, rm_ble_mesh_network_gatt_proxy_adv_mode_t proxy_adv_mode) |
| fsp_err_t(* | stopProxyServerAdv )(rm_ble_mesh_network_ctrl_t *const p_ctrl) |
| fsp_err_t(* | allocateSeqNumber )(rm_ble_mesh_network_ctrl_t *const p_ctrl, uint32_t *const p_seq_num) |
| fsp_err_t(* | getSeqNumberState )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t *const p_seq_num_state) |
| fsp_err_t(* | setSeqNumberState )(rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t const *const p_seq_num_state) |
| fsp_err_t(* | resetNetCache )(rm_ble_mesh_network_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_network_api_t::open) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_cfg_t const *const p_cfg)

Register Interface with Network Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_network_api_t::close) (rm_ble_mesh_network_ctrl_t *const p_ctrl)

Unregister Interface with Network Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ broadcastSecureBeacon

fsp_err_t(* rm_ble_mesh_network_api_t::broadcastSecureBeacon) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle)

API to send Secure Network Beacon.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Subnet handle of the network to be broadcasted. |

### ◆ sendPduOnInterface

fsp_err_t(* rm_ble_mesh_network_api_t::sendPduOnInterface) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_network_header_t const *const p_header, rm_ble_mesh_buffer_t const *const p_buffer)

Extension API to send network PDUs on selected network interfaces.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_route_info | Pointer to network configuration information. |
| [in] | p_header | Pointer to network Header. |
| [in] | p_buffer | Pointer to Lower Transport Payload. |

### ◆ getAddressType

fsp_err_t(* rm_ble_mesh_network_api_t::getAddressType) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_network_address_type_t *const p_type)

To get address type.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | Input network address. |
| [in] | p_type | One of the following address type<br>RM_BLE_MESH_NETWORK_ADDRESS_TYPE_INVALID<br>RM_BLE_MESH_NETWORK_ADDRESS_TYPE_UNICAST<br>RM_BLE_MESH_NETWORK_ADDRESS_TYPE_GROUP |

### ◆ fetchProxyState

fsp_err_t(* rm_ble_mesh_network_api_t::fetchProxyState) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_gatt_proxy_state_t *const p_proxy_state)

Check if the proxy module is ready to handle proxy messages/events.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_proxy_state | Returns the current state of the proxy. |

### ◆ setProxyFilter

fsp_err_t(* rm_ble_mesh_network_api_t::setProxyFilter) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_filter_type_t type)

Set proxy server's filter type.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_route_info | Pointer to network configuration information. |
| [in] | type | Type of the proxy filter to be set. Either RM_BLE_MESH_NETWORK_PROXY_FILTER_TYPE_WHITELIST or RM_BLE_MESH_NETWORK_PROXY_FILTER_TYPE_BLACKLIST |

### ◆ configProxyFilter

fsp_err_t(* rm_ble_mesh_network_api_t::configProxyFilter) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_route_info_t const *const p_route_info, rm_ble_mesh_proxy_config_opcode_t opcode, rm_ble_mesh_network_proxy_address_list_t *const p_addr_list)

Add or Delete/Remove addresses to/from proxy filter list.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_route_info | Pointer to network configuration information. |
| [in] | opcode | Operation to be performed. Either RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_ADD_TO_FILTER or RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_REM_FROM_FILTER |
| [in] | p_addr_list | Pointer to list of address to be added/deleted. And count of addresses present in the provided List. |

### ◆ startProxyServerAdv

fsp_err_t(* rm_ble_mesh_network_api_t::startProxyServerAdv) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, rm_ble_mesh_network_gatt_proxy_adv_mode_t proxy_adv_mode)

Start connectable advertisements for a proxy server.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Subnet handle which the proxy server is part of network. |
| [in] | proxy_adv_mode | Mode of proxy advertisements. This could be of two types RM_BLE_MESH_NETWORK_GATT_PROXY_ADV_MODE_NET_ID or RM_BLE_MESH_NETWORK_GATT_PROXY_ADV_MODE_NODE_ID |

### ◆ stopProxyServerAdv

fsp_err_t(* rm_ble_mesh_network_api_t::stopProxyServerAdv) (rm_ble_mesh_network_ctrl_t *const p_ctrl)

Stop connectable advertisements for a proxy server.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ allocateSeqNumber

fsp_err_t(* rm_ble_mesh_network_api_t::allocateSeqNumber) (rm_ble_mesh_network_ctrl_t *const p_ctrl, uint32_t *const p_seq_num)

To allocate sequence number.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_seq_num | Location where sequence number to be filled. |

### ◆ getSeqNumberState

fsp_err_t(* rm_ble_mesh_network_api_t::getSeqNumberState) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t *const p_seq_num_state)

To get current sequence number state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_seq_num_state | Location where sequence number state to be filled. |

### ◆ setSeqNumberState

fsp_err_t(* rm_ble_mesh_network_api_t::setSeqNumberState) (rm_ble_mesh_network_ctrl_t *const p_ctrl, rm_ble_mesh_network_seq_number_state_t const *const p_seq_num_state)

To set current sequence number state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_seq_num_state | Location from where sequence number state to be taken. |

### ◆ resetNetCache

fsp_err_t(* rm_ble_mesh_network_api_t::resetNetCache) (rm_ble_mesh_network_ctrl_t *const p_ctrl)

To reinitialize all Network Layer cache entries.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_ble_mesh_network_instance_t

struct rm_ble_mesh_network_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_network_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_network_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_network_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ RM_BLE_MESH_NETWORK_UNASSIGNED_ADDRESS

| #define RM_BLE_MESH_NETWORK_UNASSIGNED_ADDRESS |
| --- |
| Unassigned Address. |

### ◆ RM_BLE_MESH_NETWORK_PRIMARY_SUBNET

| #define RM_BLE_MESH_NETWORK_PRIMARY_SUBNET |
| --- |
| Primary Subnet - NetKey Index is 0x000 |

### ◆ RM_BLE_MESH_NETWORK_INVALID_SUBNET_HANDLE

| #define RM_BLE_MESH_NETWORK_INVALID_SUBNET_HANDLE |
| --- |
| Invalid Subnet Handle |

### ◆ RM_BLE_MESH_NETWORK_INVALID_APPKEY_HANDLE

| #define RM_BLE_MESH_NETWORK_INVALID_APPKEY_HANDLE |
| --- |
| Invalid AppKey Handle |

### ◆ RM_BLE_MESH_NETWORK_INVALID_NID

| #define RM_BLE_MESH_NETWORK_INVALID_NID |
| --- |
| Invalid NID Identifier. The NID is a 7-bit value that identifies the security material that is used to secure Network PDUs. Treating 0xFF as Invalid NID value. |

## Typedef Documentation

### ◆ rm_ble_mesh_network_address_t

| typedef uint16_t rm_ble_mesh_network_address_t |
| --- |
| Network Address Type |

### ◆ rm_ble_mesh_network_subnet_handle_t

| typedef uint16_t rm_ble_mesh_network_subnet_handle_t |
| --- |
| Subnet Handle |

### ◆ rm_ble_mesh_network_appkey_handle_t

| typedef uint16_t rm_ble_mesh_network_appkey_handle_t |
|---|
| AppKey Handle |

### ◆ rm_ble_mesh_network_proxy_address_t

| typedef uint16_t rm_ble_mesh_network_proxy_address_t |
|---|
| Proxy Address |

### ◆ rm_ble_mesh_network_interface_handle_t

| typedef uint8_t rm_ble_mesh_network_interface_handle_t |
|---|
| Network Interface Handle |

### ◆ rm_ble_mesh_network_ctrl_t

| typedef void rm_ble_mesh_network_ctrl_t |
|---|
| BLE MESH NETWORK control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_network_old_packet_treatment_t

| enum rm_ble_mesh_network_old_packet_treatment_t | |
|---|---|
| Ignore network cache wrapping. | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_OLD_PACKET_TREATMENT_DROPPED | Old packets will be dropped. |
| RM_BLE_MESH_NETWORK_OLD_PACKET_TREATMENT_PROCESSED | Ignores cache wrapping. Old packets will be processed. |

### ◆ rm_ble_mesh_network_rx_state_event_t

| enum rm_ble_mesh_network_rx_state_event_t | |
|---|---|
| Whether to enable the RX callback event or not. | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_RX_STATE_EVENT_DISABLE | RX state callback disable. |
| RM_BLE_MESH_NETWORK_RX_STATE_EVENT_ENABLE | RX state callback enable. |

### ◆ rm_ble_mesh_network_tx_state_event_t

| enum rm_ble_mesh_network_tx_state_event_t | |
|---|---|
| Whether to enable the TX callback event or not. | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_TX_STATE_EVENT_DISABLE | TX state callback disable. |
| RM_BLE_MESH_NETWORK_TX_STATE_EVENT_ENABLE | TX state callback enable. |

### ◆ rm_ble_mesh_network_address_type_t

| enum rm_ble_mesh_network_address_type_t | |
|---|---|
| Address Type | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_ADDRESS_TYPE_INVALID | Invalid address type |
| RM_BLE_MESH_NETWORK_ADDRESS_TYPE_UNICAST | Unicast address type |
| RM_BLE_MESH_NETWORK_ADDRESS_TYPE_VIRTUAL | Virtual address type |
| RM_BLE_MESH_NETWORK_ADDRESS_TYPE_GROUP | Group address type |

### ◆ rm_ble_mesh_proxy_filter_type_t

| enum rm_ble_mesh_proxy_filter_type_t | |
|---|---|
| Data structures for filter type and address list | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_PROXY_FILTER_TYPE_WHITELIST | GATT Proxy Filter Types - Whitelist |
| RM_BLE_MESH_NETWORK_PROXY_FILTER_TYPE_BLACKLIST | GATT Proxy Filter Type - Blacklist |

### ◆ rm_ble_mesh_proxy_config_opcode_t

| enum rm_ble_mesh_proxy_config_opcode_t | |
|---|---|
| GATT Proxy Configuration Opcodes | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_SET_FILTER | GATT Proxy Configuration - Set Filter Opcode |
| RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_ADD_TO_FILTER | GATT Proxy Configuration - Add to Filter Opcode |
| RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_REM_FROM_FILTER | GATT Proxy Configuration - Remove From Filter Opcode |
| RM_BLE_MESH_NETWORK_PROXY_CONFIG_OPECODE_FILTER_STATUS | GATT Proxy Configuration - Filter Status Opcode |

### ◆ rm_ble_mesh_network_gatt_proxy_state_t

enum rm_ble_mesh_network_gatt_proxy_state_t

GATT Proxy States.

| Proxy Callback | Proxy Iface | Error Code |
|---|---|---|
| NULL | Down | MS_PROXY_NULL |
| NULL | Up | MS_PROXY_NULL |
| !NULL | Down | MS_PROXY_READY |
| !NULL | UP | MS_PROXY_CONNECTED |

| Enumerator | |
|---|---|
| RM_BLE_MESH_NETWORK_GATT_PROXY_STATE_NULL | GATT Proxy State - Invalid/Not Initialized |
| RM_BLE_MESH_NEgTWORK_GATT_PROXY_STATE_READY | GATT Proxy State - Ready/Initialized |
| RM_BLE_MESH_NETWORK_GATT_PROXY_STATE_CONNECTED | GATT Proxy State - Connected |

### ◆ rm_ble_mesh_network_gatt_proxy_adv_mode_t

enum rm_ble_mesh_network_gatt_proxy_adv_mode_t

GATT Proxy ADV Modes

| Enumerator | |
|---|---|
| RM_BLE_MESH_NETWORK_GATT_PROXY_ADV_MODE_NET_ID | Network ID Type |
| RM_BLE_MESH_NETWORK_GATT_PROXY_ADV_MODE_NODE_ID | Node Identity Type |

### ◆ rm_ble_mesh_network_event_t

| enum rm_ble_mesh_network_event_t | |
|---|---|
| GATT Proxy Events | |
| Enumerator | |
| RM_BLE_MESH_NETWORK_EVENT_PROXY_UP | GATT Proxy Event - Interface Up |
| RM_BLE_MESH_NETWORK_EVENT_PROXY_DOWN | GATT Proxy Event - Interface Down |
| RM_BLE_MESH_NETWORK_EVENT_PROXY_STATUS | GATT Proxy Event - Status |
| RM_BLE_MESH_NETWORK_EVENT_RECEIVE | GATT Proxy Event - Receive |
| RM_BLE_MESH_NETWORK_EVENT_TX_QUEUE_EMPTY | GATT Proxy Event - TX queue empty |

## BLE Mesh Provision Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Provision functions.

# Summary

The BLE Mesh Provision interface for the BLE Mesh Provision (BLE MESH PROVISION) peripheral provides BLE Mesh Provision functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_provision_device_t |
| struct | rm_ble_mesh_provision_oob_info_t |
| struct | rm_ble_mesh_provision_capabilities_t |
| struct | rm_ble_mesh_provision_method_t |
| struct | rm_ble_mesh_provision_data_t |
| struct | rm_ble_mesh_provision_device_info_t |
| struct | rm_ble_mesh_provision_callback_args_t |

| | |
|---|---|
| struct | rm_ble_mesh_provision_cfg_t |
| struct | rm_ble_mesh_provision_api_t |
| struct | rm_ble_mesh_provision_instance_t |

### Macros

| | |
|---|---|
| #define | RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE |
| #define | RM_BLE_MESH_PROVISION_ECDH_KEY_SIZE |
| #define | RM_BLE_MESH_PROVISION_OOB_VALUE_SIZE |
| #define | RM_BLE_MESH_PROVISION_URI_HASH_SIZE |
| #define | RM_BLE_MESH_PROVISION_HANDLE_INVALID |

### Typedefs

| | |
|---|---|
| typedef uint8_t | rm_ble_mesh_provision_handle_t |
| typedef void | rm_ble_mesh_provision_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_provision_role_t |
| enum | rm_ble_mesh_provision_bearer_type_t |
| enum | rm_ble_mesh_provision_pdu_type_t |
| enum | rm_ble_mesh_provision_pub_key_value_t |
| enum | rm_ble_mesh_provision_auth_method_t |
| enum | rm_ble_mesh_provision_output_oob_action_t |
| enum | rm_ble_mesh_provision_input_oob_action_t |
| enum | rm_ble_mesh_provision_adv_transport_opcode_t |
| enum | rm_ble_mesh_provision_gatt_transport_opcode_t |
| enum | rm_ble_mesh_provision_error_code_t |
| enum | rm_ble_mesh_provision_link_close_reason_t |
| enum | rm_ble_mesh_provision_oob_type_t |

| enum | rm_ble_mesh_provision_event_type_t |
|---|---|

## Data Structure Documentation

### ◆ rm_ble_mesh_provision_device_t

| struct rm_ble_mesh_provision_device_t | | |
|---|---|---|
| Device Information used for Provisioning | | |
| Data Fields | | |
| uint8_t | uuid[ RM_BLE_MESH_DEVICE_UUID_SIZE] | Device UUID - Used in unprovisioned device beacon and Provisioning Invite |
| uint16_t | oob | OOB Information |
| rm_ble_mesh_buffer_t * | uri | URI if any, to be given in encoded form |

### ◆ rm_ble_mesh_provision_oob_info_t

| struct rm_ble_mesh_provision_oob_info_t | | |
|---|---|---|
| OOB type for provisioning | | |
| Data Fields | | |
| uint16_t | action | OOB Action information |
| uint8_t | size | OOB Size information |

### ◆ rm_ble_mesh_provision_capabilities_t

| struct rm_ble_mesh_provision_capabilities_t | | |
|---|---|---|
| Device capabilities used for Provisioning | | |
| Data Fields | | |
| uint8_t | num_elements | Number of Elements |
| uint16_t | supported_algorithms | Supported algorithms |
| uint8_t | supported_pubkey | Public key type |
| uint8_t | supported_soob | Static OOB type |
| rm_ble_mesh_provision_oob_info_t | output_oob | Output OOB information |
| rm_ble_mesh_provision_oob_info_t | input_oob | Input OOB information |

### ◆ rm_ble_mesh_provision_method_t

| struct rm_ble_mesh_provision_method_t |
|---|
| Provisioning method information |
| |

| Data Fields | | |
|---|---|---|
| uint8_t | algorithm | Algorithm selected |
| uint8_t | pubkey | Public key usage |
| uint8_t | auth | Authentication type |
| rm_ble_mesh_provision_oob_info_t | oob | OOB type |

### ◆ rm_ble_mesh_provision_data_t

| struct rm_ble_mesh_provision_data_t | | |
|---|---|---|
| Data exchanged during Provisioning procedure | | |
| Data Fields | | |
| uint8_t | netkey[RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE] | NetKey |
| uint16_t | keyid | Index of the NetKey |
| uint8_t | flags | Flags bit-mask bit 0: Key Refresh Flag. 0: Not-In-Phase2 1: In-Phase2 bit 1: IV Update Flag 0: Normal operation 1: IV Update active<br><br>bits 2-7: RFU |
| uint32_t | iv_index | Current value of the IV index |
| uint16_t | unicast_addr | Unicast address of the primary element |

### ◆ rm_ble_mesh_provision_device_info_t

| struct rm_ble_mesh_provision_device_info_t | | |
|---|---|---|
| Provisioning device information | | |
| Data Fields | | |
| rm_ble_mesh_provision_bearer_type_t | type | Provisioning Bearer Types. |
| rm_ble_mesh_provision_device_t * | p_device | Device Information used for Provisioning. |

### ◆ rm_ble_mesh_provision_callback_args_t

| struct rm_ble_mesh_provision_callback_args_t | | |
|---|---|---|
| Mesh model client callback parameter definition | | |
| Data Fields | | |
| rm_ble_mesh_provision_handle_t * | p_handle | Handle to reference the active provisioning context. |

| | | |
|---|---|---|
| rm_ble_mesh_provision_event_type_t | event_type | Provisioning event type. |
| rm_ble_mesh_error_code_t | event_result | BLE MESH error code. |
| rm_ble_mesh_buffer_t | event_data | Payload type. |
| void * | p_context | |

### ◆ rm_ble_mesh_provision_cfg_t

| struct rm_ble_mesh_provision_cfg_t | |
|---|---|
| BLE MESH PROVISION configuration parameters. | |
| **Data Fields** | |
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| rm_ble_mesh_provision_capabilities_t * | p_capabilities |
| | Device capabilities used for Provisioning. |
| | |
| rm_ble_mesh_instance_t const * | p_mesh_instance |
| | Instance structure of BLE Mesh. |
| | |
| void(* | p_callback )(rm_ble_mesh_provision_callback_args_t *p_args) |
| | Callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ channel

uint32_t rm_ble_mesh_provision_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

### ◆ rm_ble_mesh_provision_api_t

struct rm_ble_mesh_provision_api_t

BLE MESH PROVISION functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_provision_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | setup )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_role_t role, rm_ble_mesh_provision_device_info_t info, uint16_t timeout) |
| | |
| fsp_err_t(* | bind )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_device_info_t info, uint8_t attention, rm_ble_mesh_provision_handle_t *const p_handle) |
| | |
| fsp_err_t(* | sendPdu )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_pdu_type_t type, rm_ble_mesh_buffer_t pdu_data) |
| | |
| fsp_err_t(* | setAuthVal )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_buffer_t auth_value) |
| | |
| fsp_err_t(* | abort )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_link_close_reason_t reason) |
| | |
| fsp_err_t(* | getLocalPublicKey )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const public_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | setLocalPublicKey )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const public_key) |
| | |
| fsp_err_t(* | generateRandomizedNumber )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_key) |
| | |
| fsp_err_t(* | setOobPublicKey )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_key, uint8_t size) |
| | |
| fsp_err_t(* | setOobAuthInfo )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_auth_info, uint8_t size) |
| | |
| fsp_err_t(* | generateEcdhKey )(rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_public_key) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_provision_api_t::open) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_cfg_t const *const p_cfg)

Open access middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_provision_api_t::close) (rm_ble_mesh_provision_ctrl_t *const p_ctrl)

Close access middleware.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ setup

fsp_err_t(* rm_ble_mesh_provision_api_t::setup) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_role_t role, rm_ble_mesh_provision_device_info_t info, uint16_t timeout)

Setup the device for provisioning.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | role | Provisioning role to be setup - Device or Provisioner. |
| [in] | info | Device information. |
| [in] | timeout | The time period for which the setup shall be active. |

### ◆ bind

fsp_err_t(* rm_ble_mesh_provision_api_t::bind) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_device_info_t info, uint8_t attention, rm_ble_mesh_provision_handle_t *const p_handle)

Bind to the peer device for provisioning

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | info | Device information. |
| [in] | attention | Attention duration. |
| [out] | p_handle | Pointer to handle. |

***Note***

*This API is for use by the Provisioner application only upon reception of an Unprovisioned Device Beacon.*

### ◆ sendPdu

fsp_err_t(* rm_ble_mesh_provision_api_t::sendPdu) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_pdu_type_t type, rm_ble_mesh_buffer_t pdu_data)

Send provisioning PDUs to the peer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_handle | Pointer to provisioning context to be used. |
| [in] | type | Following PDU types are handled - RM_BLE_MESH_PROVISION_PDU_TYPE_START RM_BLE_MESH_PROVISION_PDU_TYPE_INPUT_COMPLETE RM_BLE_MESH_PROVISION_PDU_TYPE_DATA |
| [in] | pdu_data | Pointer to the data corresponding to the above PDUs. |

### ◆ setAuthVal

fsp_err_t(* rm_ble_mesh_provision_api_t::setAuthVal) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_buffer_t auth_value)

Set the display Auth-Value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_handle | Pointer to provisioning context to be used. |
| [in] | auth_value | Pointer to the Authval (UINT32 *) or (uint8_t *). |

### ◆ abort

fsp_err_t(* rm_ble_mesh_provision_api_t::abort) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, rm_ble_mesh_provision_handle_t const *const p_handle, rm_ble_mesh_provision_link_close_reason_t reason)

Abort the provisioning procedure

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_handle | Pointer to provisioning context to be used. |
| [in] | reason | Reason for termination. |

### ◆ getLocalPublicKey

fsp_err_t(* rm_ble_mesh_provision_api_t::getLocalPublicKey) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const public_key)

Utility API to get current ECDH Public Key to be used for Provisioning

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | public_key | To a pointer of uint8_t array of length RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE. |

### ◆ setLocalPublicKey

fsp_err_t(* rm_ble_mesh_provision_api_t::setLocalPublicKey) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const public_key)

Utility API to set current ECDH Public Key to be used for Provisioning

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | public_key | To a pointer of uint8_t array of length RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE. |

### ◆ generateRandomizedNumber

fsp_err_t(* rm_ble_mesh_provision_api_t::generateRandomizedNumber) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_key)

Utility API to generate 128bits (16 bytes) randomized number to be used for provisioning.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_key | Pointer to buffer to store random number. |

### ◆ setOobPublicKey

fsp_err_t(* rm_ble_mesh_provision_api_t::setOobPublicKey) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_key, uint8_t size)

Utility API to set device out of band public key for provisioning.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_key | Pointer to public key. |
| [in] | size | Size of public key. |

### ◆ setOobAuthInfo

fsp_err_t(* rm_ble_mesh_provision_api_t::setOobAuthInfo) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t const *const p_auth_info, uint8_t size)

Utility API to set device out of band authentication information for provisioning.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_auth_info | Pointer to authentication information. |
| [in] | size | Size of authentication information. |

### ◆ generateEcdhKey

fsp_err_t(* rm_ble_mesh_provision_api_t::generateEcdhKey) (rm_ble_mesh_provision_ctrl_t *const p_ctrl, uint8_t *const p_public_key)

Utility API to generate ECDH Public Key to be used for Provisioning

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_public_key | Pointer to public key. Size of public key is RM_BLE_MESH_PROVISION_ECDH_KEY_SIZE. |

### ◆ rm_ble_mesh_provision_instance_t

struct rm_ble_mesh_provision_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_provision_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_provision_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_provision_api_t const * | p_api | Pointer to the API structure for this instance. |

## Macro Definition Documentation

### ◆ RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE

#define RM_BLE_MESH_PROVISION_KEY_NETKEY_SIZE

Provisioning array size requirements Provisioning key NetKey size

### ◆ RM_BLE_MESH_PROVISION_ECDH_KEY_SIZE

#define RM_BLE_MESH_PROVISION_ECDH_KEY_SIZE

Provisioning ECDH Key size

### ◆ RM_BLE_MESH_PROVISION_OOB_VALUE_SIZE

#define RM_BLE_MESH_PROVISION_OOB_VALUE_SIZE

Provisioning OOB value size

### ◆ RM_BLE_MESH_PROVISION_URI_HASH_SIZE

| #define RM_BLE_MESH_PROVISION_URI_HASH_SIZE |
| --- |
| Provisioning URI hash size |

### ◆ RM_BLE_MESH_PROVISION_HANDLE_INVALID

| #define RM_BLE_MESH_PROVISION_HANDLE_INVALID |
| --- |
| Invalid Provisioning Handle |

## Typedef Documentation

### ◆ rm_ble_mesh_provision_handle_t

| typedef uint8_t rm_ble_mesh_provision_handle_t |
| --- |
| Handle to reference the active provisioning context |

### ◆ rm_ble_mesh_provision_ctrl_t

| typedef void rm_ble_mesh_provision_ctrl_t |
| --- |
| BLE MESH PROVISION control block. Allocate an instance specific control block to pass into the BLE MESH API calls. |

## Enumeration Type Documentation

### ◆ rm_ble_mesh_provision_role_t

| enum rm_ble_mesh_provision_role_t | |
| --- | --- |
| Provisioning Roles | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_ROLE_DEVICE | Device role. |
| RM_BLE_MESH_PROVISION_ROLE_PROVISIONER | Provisioner role. |

### ◆ rm_ble_mesh_provision_bearer_type_t

| enum rm_ble_mesh_provision_bearer_type_t | |
|---|---|
| Provisioning Bearer Types | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_BEARER_TYPE_ADV | Advertising bearer type. |
| RM_BLE_MESH_PROVISION_BEARER_TYPE_GATT | GATT bearer type. |

### ◆ rm_ble_mesh_provision_pdu_type_t

| enum rm_ble_mesh_provision_pdu_type_t | |
|---|---|
| Provisioning PDU Types | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_PDU_TYPE_INVITE | Invite PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_CAPAB | Capable PDu type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_START | Start PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_PUBKEY | Public key PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_INPUT_COMPLETE | Input complete PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_CONF | Configuration PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_RAND | Random PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_DATA | Data PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_COMPLETE | Complete PDU type. |
| RM_BLE_MESH_PROVISION_PDU_TYPE_FAILED | Failed PDU type. |

### ◆ rm_ble_mesh_provision_pub_key_value_t

| enum rm_ble_mesh_provision_pub_key_value_t | |
|---|---|
| Provisioning public key values | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_PUBKEY_NO_OOB | Public key no OOB. |
| RM_BLE_MESH_PROVISION_PUBKEY_OOB | Public key OOB. |

### ◆ rm_ble_mesh_provision_auth_method_t

| enum rm_ble_mesh_provision_auth_method_t | |
|---|---|
| Provisioning authentication method values | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_AUTH_METHOD_OOB_NONE | Authentication method none. |
| RM_BLE_MESH_PROVISION_AUTH_METHOD_OOB_STATIC | Authentication method static. |
| RM_BLE_MESH_PROVISION_AUTH_METHOD_OOB_OUTPUT | Authentication method output. |
| RM_BLE_MESH_PROVISION_AUTH_METHOD_OOB_INPUT | Authentication method input. |

### ◆ rm_ble_mesh_provision_output_oob_action_t

| enum rm_ble_mesh_provision_output_oob_action_t | |
|---|---|
| Provisioning Output OOB action values | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_OUTPUT_OOB_ACTION_BLINK | Output OOB action blink. |
| RM_BLE_MESH_PROVISION_OUTPUT_OOB_ACTION_BEEP | Output OOB action beep. |
| RM_BLE_MESH_PROVISION_OUTPUT_OOB_ACTION_VIBRATE | Output OOB action vibrate. |
| RM_BLE_MESH_PROVISION_OUTPUT_OOB_ACTION_NUMERIC | Output OOB action numeric. |
| RM_BLE_MESH_PROVISION_OUTPUT_OOB_ACTION_ALPHANUMERIC | Output OOB action alphanumeric. |

### ◆ rm_ble_mesh_provision_input_oob_action_t

| enum rm_ble_mesh_provision_input_oob_action_t | |
|---|---|
| Provisioning Input OOB action values | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_INPUT_OOB_ACTION_PUSH | Input OOB action push. |
| RM_BLE_MESH_PROVISION_INPUT_OOB_ACTION_TWIST | Input OOB action twist. |
| RM_BLE_MESH_PROVISION_INPUT_OOB_ACTION_NUMERIC | Input OOB action numeric. |
| RM_BLE_MESH_PROVISION_INPUT_OOB_ACTION_ALPHANUMERIC | Input OOB action alphanumeric. |

### ◆ rm_ble_mesh_provision_adv_transport_opcode_t

| enum rm_ble_mesh_provision_adv_transport_opcode_t | |
|---|---|
| Specification defined transport Opcodes for PB-ADV bearer | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_ADV_TRANSPORT_OPECODE_OPEN_REQ | Link Open Request |
| RM_BLE_MESH_PROVISION_ADV_TRANSPORT_OPECODE_OPEN_CNF | Link Open Confirm |
| RM_BLE_MESH_PROVISION_ADV_TRANSPORT_OPECODE_CLOSE_IND | Link Close Indication |

### ◆ rm_ble_mesh_provision_gatt_transport_opcode_t

| enum rm_ble_mesh_provision_gatt_transport_opcode_t | |
|---|---|
| Implementation specific transport Opcodes for PB-GATT bearer | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_GATT_TRANSPORT_OPECODE_OPEN_IND | Link Open Indication |
| RM_BLE_MESH_PROVISION_GATT_TRANSPORT_OPECODE_CLOSE_IND | Link Close Indication |

### ◆ rm_ble_mesh_provision_error_code_t

| enum rm_ble_mesh_provision_error_code_t | |
|---|---|
| Provisioning Failure Error Codes | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_ERROR_CODE_PROHIBITED | Failure error code prohibited. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_INVALID_PDU | Failure error code invalid PDU. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_INVALID_FORMAT | Failure error code invalid format. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_UNEXPECTED_PDU | Failure error code unexpected PDU. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_CONFIRMATION_FAILED | Failure error code confirmation failed. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_OUT_OF_RESOURCES | Failure error code out of resources. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_DECRYPTION_FAILED | Failure error code decryption failed. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_UNEXPECTED_ERROR | Failure error code unexpected error. |
| RM_BLE_MESH_PROVISION_ERROR_CODE_CANNOT_ASSIGN_ADDRESS | Failure error code cannot assign address. |

### ◆ rm_ble_mesh_provision_link_close_reason_t

| enum rm_ble_mesh_provision_link_close_reason_t | |
|---|---|
| Provisioning LinkClose Error codes | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_LINK_CLOSE_REASON_SUCCESS | Link close error code reason success. |
| RM_BLE_MESH_PROVISION_LINK_CLOSE_REASON_TIMEOUT | Link close error code reason timeout. |
| RM_BLE_MESH_PROVISION_LINK_CLOSE_REASON_FAIL | Link close error code reason fail. |

### ◆ rm_ble_mesh_provision_oob_type_t

| enum rm_ble_mesh_provision_oob_type_t | |
|---|---|
| Provisioning OOB type masks for ADV data | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_OOB_TYPE_OTHER | Other OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_URI | URI OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_2DMRC | 2DMRC OOB type mask |
| RM_BLE_MESH_PROVISION_OOB_TYPE_BARCODE | Bar code OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_NFC | NFC OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_NUMBER | Number OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_STRING | String OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_ONBOX | On box OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_INSIDEBOX | Inside box OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_ONPIECEOFPAPER | On piece of paper OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_INSIDEMANUAL | Inside manual OOB type mask. |
| RM_BLE_MESH_PROVISION_OOB_TYPE_ONDEVICE | On device OOB type mask. |

### ◆ rm_ble_mesh_provision_event_type_t

| enum rm_ble_mesh_provision_event_type_t | |
|---|---|
| Provision Events The Asynchronous Events notified to Application by the Module. | |
| Enumerator | |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_UNPROVISIONED_BEACON | This event indicates the availability of an unprovisioned device beacon, with the following values as parameters in the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters<br>[in] phandle Pointer to |

| | | | | |
|---|---|---|---|---|
| | | | | the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISION_EVENT_TYPE_UNPROVISIONED_BEACON. |
| | | [in] | event_result | |
| | | [in] | event_data | Pointer to the array with the UUID of the device. |
| | | [in] | event_datalen | RM_BLE_MESH_DEVICE_UUID_SIZE |
| | *Note* *This event is received by the Provisioner application. On reception of this event, the application shall make use of the RM_BLE_MESH_PROVISION_Bind() to initiate the provisioning procedure.* | | | |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVISIONING_SETUP | This event indicates that the provisioning procedure capability exchange is complete, with the following values as parameters in the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters | | | |
| | | [in] | phandle | Pointer to the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVISIONING_SETUP. |
| | | [in] | event_result | FSP_SUCCESS on successful |

| | | | | |
|---|---|---|---|---|
| | | | | procedure completion, else an Error Code. |
| | | [in] | event_data | When local provisioner, this contains peer device capabilities and when local device, this contains the attention timeout value. |
| | | [in] | event_datalen | When local provisioner, sizeof( rm_ble_mesh_provision_capabilities_t) and when local device, size of(uint32_t). |
| | *Note* *When local provisioner, the appliation shall select the required capability from the received capabilities and choose to start the procedure by calling RM_BLE_MESH_PROVISION_SendPdu() put RM_BLE_MESH_PROVISION_EVENT_TYPE _PROVISIONING_SETUP as the third argument.* | | | |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_OOB_DISPLAY | This event indicates to the application the OOB random data that is to be displayed on the UI via the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters | | | |
| | | [in] | phandle | Pointer to the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISI |

| | | | | |
|---|---|---|---|---|
| | | | | ON_EVENT_TYPE_OOB_DISPLAY. |
| | | [in] | event_result | FSP_SUCCESS on successful procedure completion, else an Error Code. |
| | | [in] | event_data | Pointer to OOB information as in rm_ble_mesh_provision_oob_info_t. |
| | | [in] | event_datalen | sizeof ( rm_ble_mesh_provision_oob_info_t ). |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_OOB_ENTRY | This event indicates to the application requesting for OOB random data that is to be used in the procedure via the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters | | | |
| | | [in] | phandle | Pointer to the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISION_EVENT_TYPE_OOB_ENTRY. |
| | | [in] | event_result | FSP_SUCCESS on successful procedure completion, else an Error Code. |
| | | [in] | event_data | Pointer to |

| | | | | |
|---|---|---|---|---|
| | | | | OOB information as in rm_ble_mesh_provision_oob_info_t. |
| | | [in] | event_datalen | sizeof ( rm_ble_mesh_provision_oob_info_t ). |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_DEVINPUT_COMPLETE | This event indicates to the application that the peer device has completed the Input of OOB when this capability is negotiated via the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters | | | |
| | | [in] | phandle | Pointer to the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISION_EVENT_TYPE_DEVINPUT_COMPLETE. |
| | | [in] | event_result | FSP_SUCCESS on successful procedure completion, else an Error Code. |
| | | [in] | event_data | NULL. |
| | | [in] | event_datalen | 0. |
| | *Note*<br>*This event is generated only for the provisioner application.* | | | |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVDATA_INFO_REQ | This event indicates to the application requesting for Provisional data to be sent to the peer device via the | | | |

| | |
|---|---|
| | rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters<br>[in] phandle — Pointer to the Provisioning context handle<br>[in] event_type — RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVDATA_INFO_REQ.<br>[in] event_result — FSP_SUCCESS on successful procedure completion, else an Error Code.<br>[in] event_data — NULL.<br>[in] event_datalen — 0.<br><br>*Note*<br>*This event is generated only for the provisioner application.* |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVDATA_INFO | This event indicates to the application the Provisional data received from the Provisioner via the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters<br>[in] phandle — Pointer to the Provisioning context handle<br>[in] event_type — RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVDATA_INFO.<br>[in] event_result — FSP_SUCCESS on successful |

| | | | | |
|---|---|---|---|---|
| | | | | procedure completion, else an Error Code. |
| | | [in] | event_data | Pointer to the provisioning data rm_ble_mesh_provision_data_t. |
| | | [in] | event_datalen | sizeof( rm_ble_mesh_provision_data_t). |
| | *Note* *This event is generated only for the device application.* | | | |
| RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVISIONING_COMPLETE | This event indicates to the application that the provisioning procedure is complete via the rm_ble_mesh_provision_callback_args_t callback.<br><br>Parameters | | | |
| | | [in] | phandle | Pointer to the Provisioning context handle |
| | | [in] | event_type | RM_BLE_MESH_PROVISION_EVENT_TYPE_PROVISIONING_COMPLETE. |
| | | [in] | event_result | FSP_SUCCESS on successful procedure completion, else an Error Code. |
| | | [in] | event_data | NULL. |
| | | [in] | event_datalen | 0. |

## BLE Mesh Scene Server Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Model Scene Server functions.

# Summary

The BLE Mesh interface for the BLE Mesh Model Scene Server (BLE MESH HEALTH SERVER) middleware provides BLE Mesh Model Scene Server functionality.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_scene_server_callback_args_t |
| struct | rm_ble_mesh_scene_server_timeout_callback_args_t |
| struct | rm_ble_mesh_scene_server_cfg_t |
| struct | rm_ble_mesh_scene_server_api_t |
| struct | rm_ble_mesh_scene_server_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ble_mesh_scene_server_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_scene_srv_event_t |

### Data Structure Documentation

#### ◆ rm_ble_mesh_scene_server_callback_args_t

| struct rm_ble_mesh_scene_server_callback_args_t | | |
|---|---|---|
| Mesh model scene server callback parameter definition | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_handle_t * | p_handle | Access Model handle. |
| rm_ble_mesh_scene_srv_event_t | event_type | Scene event types. |
| | | |

| | | |
|---|---|---|
| uint8_t * | p_event_data | Pointer to event data. |
| uint16_t | event_data_length | Event data length. |

### ◆ rm_ble_mesh_scene_server_timeout_callback_args_t

struct rm_ble_mesh_scene_server_timeout_callback_args_t

Mesh model scene server publication timeout callback.

Access Layer calls the registered callback to indicate Publication Timeout for the associated model.

**Parameters**

| | |
|---|---|
| p_context | User data. |
| p_handle | Model Handle. |
| p_blob | Blob if any or NULL. |

Data Fields

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |
| rm_ble_mesh_access_model_handle_t * | p_handle | Access Model handle. |
| void * | p_blob | Binary Large Object. |

### ◆ rm_ble_mesh_scene_server_cfg_t

struct rm_ble_mesh_scene_server_cfg_t

BLE mesh model scene server configuration parameters.

**Data Fields**

| | |
|---|---|
| rm_ble_mesh_access_instance_t const * | p_access_instance |
| | Access Layer instance structure. More... |
| | |
| rm_ble_mesh_access_model_handle_t | model_handle |
| | Access Model handle. |
| | |
| rm_ble_mesh_access_model_handle_t | setup_server_handle |
| | Access Model handle for setup server. |
| | |
| void(* | p_callback )(rm_ble_mesh_scene_server_callback_args_t *p_args) |

| | |
|---|---|
| | Mesh model scene server callback. |
| | |
| void(* | p_timeout_callback )(rm_ble_mesh_scene_server_timeout_callback_args_t *p_args) |
| | Mesh model scene server publication timeout callback. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_access_instance

rm_ble_mesh_access_instance_t const* rm_ble_mesh_scene_server_cfg_t::p_access_instance

Access Layer instance structure.

the parameters for initialization.

### ◆ rm_ble_mesh_scene_server_api_t

struct rm_ble_mesh_scene_server_api_t

Shared Interface definition for BLE MESH

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_scene_server_ctrl_t *const p_ctrl, rm_ble_mesh_scene_server_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_scene_server_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stateUpdate )(rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_scene_server_api_t::open) (rm_ble_mesh_scene_server_ctrl_t *const p_ctrl, rm_ble_mesh_scene_server_cfg_t const *const p_cfg)

API to open scene server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_scene_server_api_t::close) (rm_ble_mesh_scene_server_ctrl_t *const p_ctrl)

API to close scene server model.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ stateUpdate

fsp_err_t(* rm_ble_mesh_scene_server_api_t::stateUpdate) (rm_ble_mesh_access_ctrl_t *const p_ctrl, rm_ble_mesh_access_server_state_t const *const p_state)

API to send reply or to update state change.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_state | Pointer to model specific current/target state parameters. |

### ◆ rm_ble_mesh_scene_server_instance_t

struct rm_ble_mesh_scene_server_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_scene_server_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_scene_server_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_scene_server_api_t | p_api | Pointer to the API structure for this instance. |

| | | |
|---|---|---|
| const * | | |

### Typedef Documentation

#### ◆ rm_ble_mesh_scene_server_ctrl_t

| typedef void rm_ble_mesh_scene_server_ctrl_t |
|---|
| BLE MESH SCENE SERVER control block. Allocate an instance specific control block to pass into the BLE mesh model scene server API calls. |

### Enumeration Type Documentation

#### ◆ rm_ble_mesh_scene_srv_event_t

| enum rm_ble_mesh_scene_srv_event_t | |
|---|---|
| Scene Event Types | |
| Enumerator | |
| RM_BLE_MESH_SCENE_SRV_EVENT_STORE | Scene Event - Store |
| RM_BLE_MESH_SCENE_SRV_EVENT_DELETE | Scene Event - Delete |
| RM_BLE_MESH_SCENE_SRV_EVENT_RECALL_START | Scene Event - Recall Start |
| RM_BLE_MESH_SCENE_SRV_EVENT_RECALL_COMPLETE | Scene Event - Recall Complete |
| RM_BLE_MESH_SCENE_SRV_EVENT_RECALL_IMMEDIATE | Scene Event - Recall Immediate |

## BLE Mesh Upper Trans Interface

Interfaces » Networking » BLE Mesh Network Interfaces

### Detailed Description

Interface for BLE Mesh Upper Trans functions.

# Summary

The BLE Mesh Upper Trans middleware provides a high-level transfer interface for BLE Mesh services.

### Data Structures

| | |
|---|---|
| struct | rm_ble_mesh_upper_trans_access_layer_pdu_t |
| struct | rm_ble_mesh_upper_trans_control_pdu_t |
| struct | rm_ble_mesh_upper_trans_friendship_setting_t |
| struct | rm_ble_mesh_upper_trans_friendship_info_t |
| struct | rm_ble_mesh_upper_trans_heartbeat_publication_info_t |
| struct | rm_ble_mesh_upper_trans_heartbeat_subscription_info_t |
| struct | rm_ble_mesh_upper_trans_callback_args_t |
| struct | rm_ble_mesh_upper_trans_cfg_t |
| struct | rm_ble_mesh_upper_trans_api_t |
| struct | rm_ble_mesh_upper_trans_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_ble_mesh_upper_trans_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_ble_mesh_upper_trans_access_message_evnet_t |
| enum | rm_ble_mesh_upper_trans_control_message_evnet_t |
| enum | rm_ble_mesh_upper_trans_control_opcode_t |
| enum | rm_ble_mesh_upper_trans_friend_role_t |
| enum | rm_ble_mesh_upper_trans_event_t |

## Data Structure Documentation

### ◆ rm_ble_mesh_upper_trans_access_layer_pdu_t

| struct rm_ble_mesh_upper_trans_access_layer_pdu_t | | |
|---|---|---|
| Access PDU | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | src_addr | Source address. |
| rm_ble_mesh_network_address_t | dst_addr | Destination address. |
| uint8_t * | p_label | Label UUID, representing Virtual |

| | | |
|---|---|---|
| | | Address of Destination. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Handle identifying the subnet. |
| rm_ble_mesh_network_appkey_handle_t | appkey_handle | Handle identifying the AppKey to be used for Transport Layer encryption. |
| uint8_t | ttl | Time to Live. |
| void * | p_parameter | Transport parameter, based on the type and header. |

### ◆ rm_ble_mesh_upper_trans_control_pdu_t

| struct rm_ble_mesh_upper_trans_control_pdu_t | | |
|---|---|---|
| Control PDU | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | src_addr | Source address. |
| rm_ble_mesh_network_address_t | dst_addr | Destination address. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Handle identifying the subnet. |
| uint8_t | ttl | Time to Live. |
| rm_ble_mesh_upper_trans_control_opcode_t | opcode | Control Packet Opcode. |
| void * | p_parameter | Transport parameter, based on the type and header. |

### ◆ rm_ble_mesh_upper_trans_friendship_setting_t

| struct rm_ble_mesh_upper_trans_friendship_setting_t | | |
|---|---|---|
| Low Power Node setting | | |
| Data Fields | | |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | The subnet to initiate the friendship procedure |
| uint8_t | criteria | Friend criteria that is required. RSSI, Receive Window, MessageQueue size requirements can be established. |
| uint8_t | rx_delay | Receive delay in milliseconds that the LPN will wait before listening to response for any request. |
| uint32_t | poll_timeout | Timeout in milliseconds after |

| | | |
|---|---|---|
| | | which the LPN will send Poll PDU to check for data from the friend. |
| uint32_t | setup_timeout | Timeout in milliseconds for which the Friend Establishment procedure is to be tried. |

### ◆ rm_ble_mesh_upper_trans_friendship_info_t

| struct rm_ble_mesh_upper_trans_friendship_info_t | | |
|---|---|---|
| Low Power Node element information | | |
| Data Fields | | |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Main subnet handle of the element |
| rm_ble_mesh_network_address_t | addr | Peer LPN/Friend Address |
| uint16_t | lpn_counter | Low Power Node Counter |
| uint16_t | friend_counter | Friend Counter |

### ◆ rm_ble_mesh_upper_trans_heartbeat_publication_info_t

| struct rm_ble_mesh_upper_trans_heartbeat_publication_info_t | | |
|---|---|---|
| Hearbeat Publication state | | |
| Data Fields | | |
| rm_ble_mesh_network_address_t | daddr | Destination address for heartbeat messages |
| uint8_t | count_log | Count to control the number of periodic heartbeat transport messages to be sent |
| uint8_t | period_log | Period to control the cadence of periodic heartbeat transport messages |
| uint8_t | ttl | TTL value to be used when sending heartbeat messages |
| uint16_t | features | Features that trigger sending heartbeat messages when changed |
| uint16_t | netkey_index | Global NetKey index of the NetKey to be used to send heartbeat messages |

### ◆ rm_ble_mesh_upper_trans_heartbeat_subscription_info_t

| struct rm_ble_mesh_upper_trans_heartbeat_subscription_info_t |
|---|
| Hearbeat Subscription state |

| Data Fields | | |
|---|---|---|
| rm_ble_mesh_network_address_t | saddr | Source address for heartbeat messages that a node shall process |
| rm_ble_mesh_network_address_t | daddr | Destination address for heartbeat messages |
| uint8_t | count_log | Counter that tracks the number of periodic heartbeat transport message received since receiving the most recent configure heartbeat Subscription Set message |
| uint8_t | period_log | Period that controls the period for processing periodical heartbeat transport control messages |
| uint16_t | min_hops | Minimum hops value registered when receiving heartbeat messages since receiving the most recent configure heartbeat Subscription Set message |
| uint16_t | max_hops | Maximum hops value registered when receiving heartbeat messages since receiving the most recent configure heartbeat Subscription Set message |

### ◆ rm_ble_mesh_upper_trans_callback_args_t

| struct rm_ble_mesh_upper_trans_callback_args_t | | |
|---|---|---|
| BLE Mesh Network callback parameter definition | | |
| Data Fields | | |
| rm_ble_mesh_upper_trans_event_t | event | Event code. |
| rm_ble_mesh_network_header_t * | p_header | Event code. |
| rm_ble_mesh_network_subnet_handle_t | subnet_handle | Associated Subnet Handle. |
| rm_ble_mesh_network_appkey_handle_t | appkey_handle | Associated AppKey Handle. |
| rm_ble_mesh_error_code_t | result | Event result. |
| rm_ble_mesh_buffer_t | event_data | |
| void * | p_context | Context provided to user during |

| | | |
|---|---|---|
| | | callback. |

### ◆ rm_ble_mesh_upper_trans_cfg_t

| struct rm_ble_mesh_upper_trans_cfg_t | |
|---|---|
| BLE MESH configuration parameters. | |
| **Data Fields** | |
| uint32_t | channel |
| | Select a channel corresponding to the channel number of the hardware. More... |
| | |
| rm_ble_mesh_upper_trans_control_message_evnet_t | control_message_evnet |
| | Whether to enable the control message event or not. |
| | |
| rm_ble_mesh_upper_trans_access_message_evnet_t | access_message_evnet |
| | Whether to enable the access message event or not. |
| | |
| rm_ble_mesh_lower_trans_instance_t const * | p_mesh_lower_trans_instance |
| | Instance structure of BLE Mesh Bearer. |
| | |
| void(* | p_callback )(rm_ble_mesh_upper_trans_callback_args_t *p_args) |
| | Callback. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t. |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ channel

uint32_t rm_ble_mesh_upper_trans_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

### ◆ rm_ble_mesh_upper_trans_api_t

struct rm_ble_mesh_upper_trans_api_t

BLE MESH functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | sendAccessPdu )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_access_layer_pdu_t const *const p_access_layer_pdu, rm_ble_mesh_lower_trans_reliable_t reliable) |
| | |
| fsp_err_t(* | sendControlPdu )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_pdu_t const *const p_control_pdu) |
| | |
| fsp_err_t(* | lpnSetupFriendship )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friendship_setting_t const *const p_setting) |
| | |
| fsp_err_t(* | lpnClearFriendship )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | lpnManageSubscription )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_opcode_t action, uint16_t const *const p_addr_list, uint16_t count) |
| | |
| fsp_err_t(* | lpnPoll )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | isValidLpnElementAddress )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle) |
| | |

| | |
|---|---|
| fsp_err_t(* | isValidLpnSubscriptionAddress )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle) |
| fsp_err_t(* | getLpnPolltimeout )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t lpn_addr, uint32_t *const p_poll_timeout) |
| fsp_err_t(* | getFriendshipInfo )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friend_role_t role, uint16_t lpn_index, rm_ble_mesh_upper_trans_friendship_info_t *const p_node) |
| fsp_err_t(* | lpnRegisterSecurityUpdate )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t flag, uint32_t ivindex) |
| fsp_err_t(* | clearAllLpn )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl) |
| fsp_err_t(* | setHeartbeatPublication )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info) |
| fsp_err_t(* | getHeartbeatPublication )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info) |
| fsp_err_t(* | setHeartbeatSubscription )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info) |
| fsp_err_t(* | getHeartbeatSubscription )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info) |
| fsp_err_t(* | triggerHeartbeat )(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, uint8_t change_in_feature_bit) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::open) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_cfg_t const *const p_cfg)

Register interface with Transport Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::close) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl)

Unregister interface with Transport Layer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ sendAccessPdu

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::sendAccessPdu) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_access_layer_pdu_t const *const p_access_layer_pdu, rm_ble_mesh_lower_trans_reliable_t reliable)

API to send Access Layer PDUs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_access_layer_pdu | Pointer to Access Layer PDUs. |
| [in] | reliable | If requires lower transport ACK, set reliable as RM_BLE_MESH_LOWER_TRANS_RELIABLE_ENABLE. |

### ◆ sendControlPdu

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::sendControlPdu) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_pdu_t const *const p_control_pdu)

API to send transport Control PDUs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_control_pdu | Pointer to control PDUs. |

### ◆ lpnSetupFriendship

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::lpnSetupFriendship) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friendship_setting_t const *const p_setting)

API to setup Friendship.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_setting | Pointer to friendship settings. |

### ◆ lpnClearFriendship

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::lpnClearFriendship) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl)

API to terminate friendship.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ lpnManageSubscription

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::lpnManageSubscription) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_control_opcode_t action, uint16_t const *const p_addr_list, uint16_t count)

API to manage friend subscription list.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | action | Will be one of RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_SUBSCRN_LIST_ADD or RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_SUBSCRN_LIST_REMOVE. |
| [in] | p_addr_list | Pointer to the packed list of addresses to be managed. |
| [in] | count | Number of addresses given. |

### ◆ lpnPoll

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::lpnPoll) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl)

To trigger Friend Poll from application.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ isValidLpnElementAddress

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::isValidLpnElementAddress) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle)

To check if address matches with any of the LPN.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | Unicast address to search. |
| [out] | p_lpn_handle | Pointer to an LPN Handle, which will be filled if match found. |

### ◆ isValidLpnSubscriptionAddress

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::isValidLpnSubscriptionAddress) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t addr, rm_ble_mesh_lower_trans_lpn_handle_t *const p_lpn_handle)

To check if valid subscription address of an LPN to receive a packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | addr | Address to search. |
| [out] | p_lpn_handle | Pointer to an LPN Handle, which will be filled if match found. |

### ◆ getLpnPolltimeout

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::getLpnPolltimeout) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_address_t lpn_addr, uint32_t *const p_poll_timeout)

To get Poll Timeout of an LPN.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | lpn_addr | LPN address to search. |
| [out] | p_poll_timeout | Pointer to a memory where poll timeout of the LPN to be filled (if match found). |

### ◆ getFriendshipInfo

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::getFriendshipInfo) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_friend_role_t role, uint16_t lpn_index, rm_ble_mesh_upper_trans_friendship_info_t *const p_node)

To get the LPN node information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | role | Local friendship role. |
| [in] | lpn_index | Index of the LPN element. |
| [out] | p_node | Pointer to copy the information. |

### ◆ lpnRegisterSecurityUpdate

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::lpnRegisterSecurityUpdate) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_network_subnet_handle_t subnet_handle, uint8_t flag, uint32_t ivindex)

To add the security update information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | subnet_handle | Handle to identify the network. |
| [in] | flag | Flag indicating the Key Refresh and IV Update state. |
| [in] | ivindex | Current IV index of the network. |

### ◆ clearAllLpn

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::clearAllLpn) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl)

To clear information related to all LPNs.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ setHeartbeatPublication

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::setHeartbeatPublication) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info)

To set the heartbeat publication data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Heartbeat Publication information data as in rm_ble_mesh_upper_trans_heartbeat_publication_info_t. |

### ◆ getHeartbeatPublication

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::getHeartbeatPublication)
(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl,
rm_ble_mesh_upper_trans_heartbeat_publication_info_t *const p_info)

To get the heartbeat publication data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Heartbeat Publication information data as in rm_ble_mesh_upper_trans_heartbeat_publication_info_t. |

### ◆ setHeartbeatSubscription

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::setHeartbeatSubscription)
(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl,
rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info)

To set the heartbeat subscription data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Heartbeat Publication information data as in rm_ble_mesh_upper_trans_heartbeat_subscription_info_t. |

### ◆ getHeartbeatSubscription

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::getHeartbeatSubscription)
(rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl,
rm_ble_mesh_upper_trans_heartbeat_subscription_info_t *const p_info)

To get the heartbeat subscription data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Heartbeat Publication information data as in rm_ble_mesh_upper_trans_heartbeat_subscription_info_t. |

### ◆ triggerHeartbeat

fsp_err_t(* rm_ble_mesh_upper_trans_api_t::triggerHeartbeat) (rm_ble_mesh_upper_trans_ctrl_t *const p_ctrl, uint8_t change_in_feature_bit)

To trigger heartbeat send on change in feature.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | change_in_feature_bit | Bit mask of the changed feature field. |

### ◆ rm_ble_mesh_upper_trans_instance_t

struct rm_ble_mesh_upper_trans_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_ble_mesh_upper_trans_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_ble_mesh_upper_trans_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_ble_mesh_upper_trans_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_ble_mesh_upper_trans_ctrl_t

typedef void rm_ble_mesh_upper_trans_ctrl_t

BLE MESH control block. Allocate an instance specific control block to pass into the BLE MESH API calls.

## Enumeration Type Documentation

### ◆ rm_ble_mesh_upper_trans_access_message_evnet_t

enum rm_ble_mesh_upper_trans_access_message_evnet_t

Whether to enable the access message event or not.

Enumerator

| | |
|---|---|
| RM_BLE_MESH_UPPER_TRANS_ACCESS_MESSAGE_EVENT_DISABLE | Access state callback disable. |
| RM_BLE_MESH_UPPER_TRANS_ACCESS_MESSAGE_EVENT_ENABLE | Access state callback enable. |

### ◆ rm_ble_mesh_upper_trans_control_message_evnet_t

| enum rm_ble_mesh_upper_trans_control_message_evnet_t | |
|---|---|
| Whether to enable the control message event or not. | |
| Enumerator | |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_MESSAGE_EVENT_DISABLE | Control state callback disable. |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_MESSAGE_EVENT_ENABLE | Control state callback enable. |

### ◆ rm_ble_mesh_upper_trans_control_opcode_t

| enum rm_ble_mesh_upper_trans_control_opcode_t | |
|---|---|
| Tranport Layer Control Packet Opcodes<br>RFU: 0x02 - 0x0F | |
| Enumerator | |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_POLL | Sent by a Low Power node to its Friend node to request any messages that it has cached for the Low Power node |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_UPDATE | Sent by a Friend node to a Low Power node to inform it about cache and/or security updates |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_REQ | Broadcast by a Low Power node to start to find a friend |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_OFFER | Sent by a Friend node to a Low Power node to offer to become its friend |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_CLEAR | Sent to a Friend node to inform a previous friend of a Low Power node about the removal of a friendship |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_CLEAR_CNF | Sent from a previous friend to Friend node to confirm that a prior friend relationship has been removed |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_SUBSCRN_LIST_ADD | Sent to a Friend node to add one or more addresses to the Friend Subscription List |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_SUBSCRN_LIST_REMOVE | Sent to a Friend node to remove one or more addresses from the Friend Subscription List |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_FRIEND_SUBSCRN_LIST_CNF | Sent by a Friend node to confirm Friend Subscription List updates |
| RM_BLE_MESH_UPPER_TRANS_CONTROL_OPCODE_HEARTBEAT | Sent by a node to let other nodes determine topology of a Subnet |

### ◆ rm_ble_mesh_upper_trans_friend_role_t

| enum rm_ble_mesh_upper_trans_friend_role_t | |
|---|---|
| Friend role | |
| Enumerator | |
| RM_BLE_MESH_UPPER_TRANS_FRIEND_ROLE_FRIEND | Friend role. |
| RM_BLE_MESH_UPPER_TRANS_FRIEND_ROLE_LPN | LPN role. |

### ◆ rm_ble_mesh_upper_trans_event_t

| enum rm_ble_mesh_upper_trans_event_t | |
|---|---|
| Callback Event | |
| Enumerator | |
| RM_BLE_MESH_UPPER_TRANS_EVENT_ACCESS | BLE Mesh Upper Trans Event - Access. |
| RM_BLE_MESH_UPPER_TRANS_EVENT_CONTROL | BLE Mesh Upper Trans Event - Control. |
| RM_BLE_MESH_UPPER_TRANS_EVENT_FRIENDSHIP_SETUP | BLE Mesh Upper Trans Event - Friendship setup. |
| RM_BLE_MESH_UPPER_TRANS_EVENT_FRIENDSHIP_SUBSCRIPTION_LIST | BLE Mesh Upper Trans Event - Friendship subscription list. |
| RM_BLE_MESH_UPPER_TRANS_EVENT_FRIENDSHIP_CLEAR | BLE Mesh Upper Trans Event - Friendship clear. |
| RM_BLE_MESH_UPPER_TRANS_EVENT_FRIENDSHIP_TERMINATE | BLE Mesh Upper Trans Event - Friendship terminate. |

## 5.3.11.4 DA16XXX AT Command Transport Layer

Interfaces » Networking

### Detailed Description

Abstraction interface for DA16XXX AT Command functions.

# Summary

The DA16XXX AT Command Transport Layer interface provides functions for data communication and buffer handling over multiple communications interfaces.

## Data Structures

| | |
|---|---|
| struct | at_transport_da16xxx_callback_args_t |
| struct | at_transport_da16xxx_cfg_t |
| struct | at_transport_da16xxx_data_t |
| struct | at_transport_da16xxx_status_t |
| struct | at_transport_da16xxx_api_t |
| struct | at_transport_da16xxx_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | at_transport_da16xxx_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | at_transport_da16xxx_event_t |

## Data Structure Documentation

### ◆ at_transport_da16xxx_callback_args_t

| |
|---|
| struct at_transport_da16xxx_callback_args_t |
| DA16xxx middleware callback parameter definition |

### ◆ at_transport_da16xxx_cfg_t

| | |
|---|---|
| struct at_transport_da16xxx_cfg_t | |
| DA16xxx middleware configuration block | |
| **Data Fields** | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| bool(* | p_callback )(at_transport_da16xxx_callback_args_t *p_args) |

| | |
|---|---|
| | Pointer to callback function. |
| | |

### ◆ at_transport_da16xxx_data_t

| struct at_transport_da16xxx_data_t | | |
|---|---|---|
| DA16xxx data structure | | |
| Data Fields | | |
| uint8_t * | p_at_cmd_string | Pointer to ATCMD string. |
| uint32_t | at_cmd_string_length | ATCMD string length. |
| uint8_t * | p_response_buffer | Pointer to ATCMD response buffer. |
| uint32_t | response_buffer_size | ATCMD response buffer string length. |
| uint32_t | timeout_ms | ATCMD timeout in ms. |
| const char * | p_expect_code | Expected string in the ATCMD response. |
| uint32_t | comm_ch_id | Communication channel ID. |

### ◆ at_transport_da16xxx_status_t

| struct at_transport_da16xxx_status_t | | |
|---|---|---|
| DA16xxx status indicators | | |
| Data Fields | | |
| bool | open | True if driver is open. |

### ◆ at_transport_da16xxx_api_t

| struct at_transport_da16xxx_api_t | |
|---|---|
| AT Command APIs | |
| **Data Fields** | |
| fsp_err_t(* | open )(at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(at_transport_da16xxx_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | atCommandSendThreadSafe )(at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd) |
| | |
| fsp_err_t(* | atCommandSend )(at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd) |

| | |
|---|---|
| | |
| fsp_err_t(* | giveMutex )(at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag) |
| | |
| fsp_err_t(* | takeMutex )(at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag) |
| | |
| fsp_err_t(* | statusGet )(at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_status_t *p_status) |
| | |
| size_t(* | bufferRecv )(at_transport_da16xxx_ctrl_t *const p_ctrl, const char *p_data, uint32_t length, uint32_t rx_timeout) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* at_transport_da16xxx_api_t::open) (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_cfg_t const *const p_cfg)

Open at cmd instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* at_transport_da16xxx_api_t::close) (at_transport_da16xxx_ctrl_t *const p_ctrl)

Close at cmd instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ atCommandSendThreadSafe

fsp_err_t(* at_transport_da16xxx_api_t::atCommandSendThreadSafe) (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd)

at cmd send thread safe.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_at_cmd | Pointer to AT command data structure. |

### ◆ atCommandSend

fsp_err_t(* at_transport_da16xxx_api_t::atCommandSend) (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_data_t *p_at_cmd)

at cmd send.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_at_cmd | Pointer to AT command data structure. |

### ◆ giveMutex

fsp_err_t(* at_transport_da16xxx_api_t::giveMutex) (at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag)

Give the mutex.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | mutex_flag | TX/RX Flags for the mutex. |

### ◆ takeMutex

fsp_err_t(* at_transport_da16xxx_api_t::takeMutex) (at_transport_da16xxx_ctrl_t *const p_ctrl, uint32_t mutex_flag)

Take the mutex .

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | mutex_flag | TX/RX Flags for the mutex. |

### ◆ statusGet

fsp_err_t(* at_transport_da16xxx_api_t::statusGet) (at_transport_da16xxx_ctrl_t *const p_ctrl, at_transport_da16xxx_status_t *p_status)

Gets the status of the configured DA16xxx transport.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the to Transport layer instance control structure. |
| [out] | p_status | Pointer to store current status. |

### ◆ bufferRecv

size_t(* at_transport_da16xxx_api_t::bufferRecv) (at_transport_da16xxx_ctrl_t *const p_ctrl, const char *p_data, uint32_t length, uint32_t rx_timeout)

Receive data from stream buffer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to Transport layer instance control structure. |
| [in] | p_data | Pointer to data. |
| [in] | length | Data length. |
| [in] | rx_timeout | Timeout for receiving data on the buffer. |
| [in] | trigger_level | Trigger level for stream buffer. |

### ◆ at_transport_da16xxx_instance_t

struct at_transport_da16xxx_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

## Typedef Documentation

### ◆ at_transport_da16xxx_ctrl_t

typedef void at_transport_da16xxx_ctrl_t

At transport control block. Allocate an instance specific control block to pass into the Communications API calls.

### Enumeration Type Documentation

#### ◆ at_transport_da16xxx_event_t

| enum at_transport_da16xxx_event_t |
| --- |
| Event in the callback function |

### 5.3.11.5 EtherCAT SSC Interface

Interfaces » Networking

### Detailed Description

Interface for EtherCAT SSC functions.

## Summary

The EtherCAT SSC port module (rm_ethercat_ssc_port) provides an API for EtherCAT Slave Stack Code that use the EtherCAT Slave Controler peripheral.

### Data Structures

| | |
| --- | --- |
| struct | ethercat_ssc_port_callback_args_t |
| struct | ethercat_ssc_port_cfg_t |
| struct | ethercat_ssc_port_api_t |
| struct | ethercat_ssc_port_instance_t |

### Typedefs

| | |
| --- | --- |
| typedef void | ethercat_ssc_port_ctrl_t |

### Enumerations

| | |
| --- | --- |
| enum | ethercat_ssc_port_event_t |

### Data Structure Documentation

#### ◆ ethercat_ssc_port_callback_args_t

| struct ethercat_ssc_port_callback_args_t | | |
| --- | --- | --- |
| Callback function parameter data | | |
| Data Fields | | |
| ethercat_ssc_port_event_t | event | Event code. |

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. Set in ethercat_ssc_port_api_t::open function in ethercat_ssc_port_cfg_t. |

### ◆ ethercat_ssc_port_cfg_t

| struct ethercat_ssc_port_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| uint32_t | reset_hold_time |
| | PHY Reset signal hold time (ms) |
| | |
| uint32_t | reset_wait_time |
| | Wait time after PHY reset relase (us) |
| | |
| uint32_t | address_offset |
| | Offset of the PHY address. |
| | |
| IRQn_Type | common_irq |
| | EtherCAT IRQ interrupt number. |
| | |
| uint8_t | common_ipl |
| | EtherCAT interrupt priority. |
| | |
| IRQn_Type | sync0_irq |
| | EtherCAT Sync0 IRQ interrupt number. |
| | |
| uint8_t | sync0_ipl |
| | EtherCAT Sync0 interrupt priority. |
| | |
| IRQn_Type | sync1_irq |

| | |
|---|---|
| | EtherCAT Sync1 IRQ interrupt number. |
| | |
| uint8_t | sync1_ipl |
| | EtherCAT Sync1 interrupt priority. |
| | |
| void(* | p_callback )(ethercat_ssc_port_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| timer_instance_t const * | p_timer_instance |
| | Pointer to Timer instance. |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_context

void* ethercat_ssc_port_cfg_t::p_context

Placeholder for user data. Passed to the user callback in ethercat_ssc_port_callback_args_t.

### ◆ ethercat_ssc_port_api_t

struct ethercat_ssc_port_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ethercat_ssc_port_ctrl_t *const p_api_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ethercat_ssc_port_ctrl_t *const p_api_ctrl) |

## Field Documentation

### ◆ open

| fsp_err_t(* ethercat_ssc_port_api_t::open) (ethercat_ssc_port_ctrl_t *const p_api_ctrl, ethercat_ssc_port_cfg_t const *const p_cfg) |
|---|

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to EtherCAT SSC port configuration structure. |

### ◆ close

| fsp_err_t(* ethercat_ssc_port_api_t::close) (ethercat_ssc_port_ctrl_t *const p_api_ctrl) |
|---|

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_api_ctrl | Pointer to control structure. |

### ◆ ethercat_ssc_port_instance_t

| struct ethercat_ssc_port_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| ethercat_ssc_port_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ethercat_ssc_port_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ethercat_ssc_port_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ethercat_ssc_port_ctrl_t

| typedef void ethercat_ssc_port_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ ethercat_ssc_port_event_t

| enum ethercat_ssc_port_event_t | |
|---|---|
| Event code of callback function | |
| Enumerator | |
| ETHERCAT_SSC_PORT_EVENT_ESC_CAT_INTERRUPT | EtherCAT Interrupt event. |
| ETHERCAT_SSC_PORT_EVENT_ESC_SYNC0_INTERRUPT | EtherCAT Sync0 Interrupt event. |
| ETHERCAT_SSC_PORT_EVENT_ESC_SYNC1_INTERRUPT | EtherCAT Sync1 Interrupt event. |

## 5.3.11.6 Ethernet Interface

Interfaces » Networking

### Detailed Description

Interface for Ethernet functions.

# Summary

The Ethernet interface provides Ethernet functionality. The Ethernet interface supports the following features:

- Transmit/receive processing (Blocking and Non-Blocking)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support

### Data Structures

| | |
|---|---|
| struct | ether_nic_info_t |
| struct | ether_callback_args_t |
| struct | ether_cfg_t |
| struct | ether_api_t |
| struct | ether_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ether_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ether_wake_on_lan_t |
| enum | ether_flow_control_t |
| enum | ether_multicast_t |
| enum | ether_promiscuous_t |
| enum | ether_zerocopy_t |
| enum | ether_event_t |

## Data Structure Documentation

### ◆ ether_nic_info_t

| struct ether_nic_info_t | | |
|---|---|---|
| Network interface device infomation | | |
| Data Fields | | |
| uint8_t * | p_mac_address | MAC address. |

### ◆ ether_callback_args_t

| struct ether_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| ether_event_t | event | Event code. |
| uint32_t | status_ecsr | ETHERC status register for interrupt handler. |
| uint32_t | status_eesr | ETHERC/EDMAC status register for interrupt handler. |
| ether_nic_info_t * | p_nic_info | Pointer of NIC info. |
| void * | p_context | Placeholder for user data. Set in ether_api_t::open function in ether_cfg_t. |

### ◆ ether_cfg_t

| struct ether_cfg_t |
|---|
| Configuration parameters. |
| **Data Fields** |

| | |
|---|---|
| uint8_t | channel |
| | Channel. |
| | |
| ether_zerocopy_t | zerocopy |
| | Zero copy enable or disable in Read/Write function. |
| | |
| ether_multicast_t | multicast |
| | Multicast enable or disable. |
| | |
| ether_promiscuous_t | promiscuous |
| | Promiscuous mode enable or disable. |
| | |
| ether_flow_control_t | flow_control |
| | Flow control functionally enable or disable. |
| | |
| ether_padding_t | padding |
| | Padding length inserted into the received Ethernet frame. |
| | |
| uint32_t | padding_offset |
| | Offset of the padding inserted into the received Ethernet frame. |
| | |
| uint32_t | broadcast_filter |
| | Limit of the number of broadcast frames received continuously. |
| | |
| uint8_t * | p_mac_address |
| | Pointer of MAC address. |
| | |
| uint8_t | num_tx_descriptors |

| | |
|---|---|
| | Number of transmission descriptor. |
| | |
| uint8_t | num_rx_descriptors |
| | Number of receive descriptor. |
| | |
| uint8_t ** | pp_ether_buffers |
| | Transmit and receive buffer. |
| | |
| uint32_t | ether_buffer_size |
| | Size of transmit and receive buffer. |
| | |
| IRQn_Type | irq |
| | Interrupt number. |
| | |
| uint32_t | interrupt_priority |
| | Interrupt priority. |
| | |
| void(* | p_callback )(ether_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| ether_phy_instance_t const * | p_ether_phy_instance |
| | Pointer to ETHER_PHY instance. |
| | |
| void * | p_context |
| | Placeholder for user data. More... |
| | |
| void const * | p_extend |
| | |

| | |
|---|---|
| | Placeholder for user extension. |
| | |

## Field Documentation

### ◆ p_context

| |
|---|
| void* ether_cfg_t::p_context |
| Placeholder for user data.<br>Placeholder for user data. Passed to the user callback in ether_callback_args_t. |

### ◆ ether_api_t

| |
|---|
| struct ether_api_t |
| Functions implemented at the HAL layer will follow this API. |

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ether_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes) |
| | |
| fsp_err_t(* | bufferRelease )(ether_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | rxBufferUpdate )(ether_ctrl_t *const p_ctrl, void *const p_buffer) |
| | |
| fsp_err_t(* | write )(ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length) |
| | |
| fsp_err_t(* | linkProcess )(ether_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | wakeOnLANEnable )(ether_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | txStatusGet )(ether_ctrl_t *const p_ctrl, void *const p_buffer_address) |
| | |
| fsp_err_t(* | callbackSet )(ether_ctrl_t *const p_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, |

| | |
|---|---|
| | ether_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ether_api_t::open) (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ether_api_t::close) (ether_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* ether_api_t::read) (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes)

Read packet if data is available.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buffer | Pointer to where to store read data. |
| [in] | length_bytes | Number of bytes in buffer |

### ◆ bufferRelease

fsp_err_t(* ether_api_t::bufferRelease) (ether_ctrl_t *const p_ctrl)

Release rx buffer from buffer pool process in zero-copy read operation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rxBufferUpdate

fsp_err_t(* ether_api_t::rxBufferUpdate) (ether_ctrl_t *const p_ctrl, void *const p_buffer)

Update the buffer pointer in the current receive descriptor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buffer | New address to write into the rx buffer descriptor. |

### ◆ write

fsp_err_t(* ether_api_t::write) (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length)

Write packet.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_buffer | Pointer to data to write. |
| [in] | frame_length | Send ethernet frame size (without 4 bytes of CRC data size). |

### ◆ linkProcess

fsp_err_t(* ether_api_t::linkProcess) (ether_ctrl_t *const p_ctrl)

Process link.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ wakeOnLANEnable

fsp_err_t(* ether_api_t::wakeOnLANEnable) (ether_ctrl_t *const p_ctrl)

Enable magic packet detection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ txStatusGet

fsp_err_t(* ether_api_t::txStatusGet) (ether_ctrl_t *const p_ctrl, void *const p_buffer_address)

Get the address of the most recently sent buffer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_buffer_address | Pointer to the address of the most recently sent buffer. |

### ◆ callbackSet

fsp_err_t(* ether_api_t::callbackSet) (ether_ctrl_t *const p_ctrl, void(*p_callback)(ether_callback_args_t *), void *const p_context, ether_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the ETHER control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ ether_instance_t

struct ether_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| ether_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ether_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ether_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ether_ctrl_t

| typedef void ether_ctrl_t |
| --- |
| Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ ether_wake_on_lan_t

| enum ether_wake_on_lan_t | |
| --- | --- |
| Wake on LAN | |
| Enumerator | |
| ETHER_WAKE_ON_LAN_DISABLE | Disable Wake on LAN. |
| ETHER_WAKE_ON_LAN_ENABLE | Enable Wake on LAN. |

### ◆ ether_flow_control_t

| enum ether_flow_control_t | |
| --- | --- |
| Flow control functionality | |
| Enumerator | |
| ETHER_FLOW_CONTROL_DISABLE | Disable flow control functionality. |
| ETHER_FLOW_CONTROL_ENABLE | Enable flow control functionality with pause frames. |

### ◆ ether_multicast_t

| enum ether_multicast_t | |
| --- | --- |
| Multicast Filter | |
| Enumerator | |
| ETHER_MULTICAST_DISABLE | Disable reception of multicast frames. |
| ETHER_MULTICAST_ENABLE | Enable reception of multicast frames. |

### ◆ ether_promiscuous_t

| enum ether_promiscuous_t | |
|---|---|
| Promiscuous Mode | |
| Enumerator | |
| ETHER_PROMISCUOUS_DISABLE | Only receive packets with current MAC address, multicast, and broadcast. |
| ETHER_PROMISCUOUS_ENABLE | Receive all packets. |

### ◆ ether_zerocopy_t

| enum ether_zerocopy_t | |
|---|---|
| Zero copy | |
| Enumerator | |
| ETHER_ZEROCOPY_DISABLE | Disable zero copy in Read/Write function. |
| ETHER_ZEROCOPY_ENABLE | Enable zero copy in Read/Write function. |

### ◆ ether_event_t

| enum ether_event_t | |
|---|---|
| Event code of callback function | |
| Enumerator | |
| ETHER_EVENT_WAKEON_LAN | Magic packet detection event. |
| ETHER_EVENT_LINK_ON | Link up detection event. |
| ETHER_EVENT_LINK_OFF | Link down detection event. |
| ETHER_EVENT_RX_COMPLETE | Receive complete event. |
| ETHER_EVENT_RX_MESSAGE_LOST | Receive FIFO overflow or Receive descriptor is full. |
| ETHER_EVENT_TX_COMPLETE | Transmit complete event. |
| ETHER_EVENT_TX_BUFFER_EMPTY | Transmit descriptor or FIFO is empty. |
| ETHER_EVENT_TX_ABORTED | Transmit abort event. |
| ETHER_EVENT_ERR_GLOBAL | Global error has occurred. |
| ETHER_EVENT_GET_NIC_INFO | Get NIC Info. |

### 5.3.11.7 Ethernet PHY Interface

Interfaces » Networking

#### Detailed Description

Interface for Ethernet PHY functions.

# Summary

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral.

The Ethernet PHY interface supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

### Data Structures

| | |
|---|---|
| struct | ether_phy_cfg_t |
| struct | ether_phy_lsi_cfg_t |
| struct | ether_phy_api_t |
| struct | ether_phy_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ether_phy_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ether_phy_lsi_type_t |
| enum | ether_phy_flow_control_t |
| enum | ether_phy_link_speed_t |
| enum | ether_phy_mii_type_t |

## Data Structure Documentation

### ◆ ether_phy_cfg_t

| struct ether_phy_cfg_t | | |
|---|---|---|
| Configuration parameters. | | |
| Data Fields | | |
| uint8_t | channel | Channel. |
| uint8_t | phy_lsi_address | DEPRECATED Address of PHY-LSI. |
| uint32_t | phy_reset_wait_time | Wait time for PHY-LSI reboot. |
| int32_t | mii_bit_access_wait_time | Wait time for MII/RMII access. |
| ether_phy_lsi_type_t | phy_lsi_type | DEPRECATED Phy LSI type. |
| ether_phy_flow_control_t | flow_control | Flow control functionally enable or disable. |
| ether_phy_mii_type_t | mii_type | Interface type is MII or RMII. |
| void * | p_context | Placeholder for user data. Passed to the user callback in ether_phy_callback_args_t. |
| void const * | p_extend | Placeholder for user extension. |

### ◆ ether_phy_lsi_cfg_t

| struct ether_phy_lsi_cfg_t | | |
|---|---|---|
| PHY LSI configuration. | | |
| Data Fields | | |
| uint8_t | address | Address of PHY-LSI. |
| ether_phy_lsi_type_t | type | Phy LSI type. |

### ◆ ether_phy_api_t

| struct ether_phy_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(ether_phy_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | chipInit )(ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data) |
| | |
| fsp_err_t(* | write )(ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data) |
| | |
| fsp_err_t(* | startAutoNegotiate )(ether_phy_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | linkPartnerAbilityGet )(ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause) |
| | |
| fsp_err_t(* | linkStatusGet )(ether_phy_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ether_phy_api_t::open) (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* ether_phy_api_t::close) (ether_phy_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ chipInit

fsp_err_t(* ether_phy_api_t::chipInit) (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg)

Initialize PHY-LSI.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ read

fsp_err_t(* ether_phy_api_t::read) (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t *const p_data)

Read register value of PHY-LSI.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | reg_addr | Register address. |
| [out] | p_data | Pointer to the location to store read data. |

### ◆ write

fsp_err_t(* ether_phy_api_t::write) (ether_phy_ctrl_t *const p_ctrl, uint32_t reg_addr, uint32_t data)

Write data to register of PHY-LSI.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | reg_addr | Register address. |
| [in] | data | Data written to register. |

### ◆ startAutoNegotiate

fsp_err_t(* ether_phy_api_t::startAutoNegotiate) (ether_phy_ctrl_t *const p_ctrl)

Start auto negotiation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ linkPartnerAbilityGet

fsp_err_t(* ether_phy_api_t::linkPartnerAbilityGet) (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)

Get the partner ability.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_line_speed_duplex | Pointer to the location of both the line speed and the duplex. |
| [out] | p_local_pause | Pointer to the location to store the local pause bits. |
| [out] | p_partner_pause | Pointer to the location to store the partner pause bits. |

### ◆ linkStatusGet

fsp_err_t(* ether_phy_api_t::linkStatusGet) (ether_phy_ctrl_t *const p_ctrl)

Get Link status from PHY-LSI interface.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ ether_phy_instance_t

| struct ether_phy_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| ether_phy_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ether_phy_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ether_phy_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ ether_phy_ctrl_t

| typedef void ether_phy_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

### Enumeration Type Documentation

#### ◆ ether_phy_lsi_type_t

| enum ether_phy_lsi_type_t | |
|---|---|
| Phy LSI | |
| Enumerator | |
| ETHER_PHY_LSI_TYPE_DEFAULT | Select default configuration. This type dose not change Phy LSI default setting by strapping option. |
| ETHER_PHY_LSI_TYPE_KSZ8091RNB | Select configuration for KSZ8091RNB. |
| ETHER_PHY_LSI_TYPE_KSZ8041 | Select configuration for KSZ8041. |
| ETHER_PHY_LSI_TYPE_DP83620 | Select configuration for DP83620. |
| ETHER_PHY_LSI_TYPE_ICS1894 | Select configuration for ICS1894. |
| ETHER_PHY_LSI_TYPE_GPY111 | Select configuration for GPY111. |
| ETHER_PHY_LSI_TYPE_CUSTOM | Select configuration for User custom. |

### ◆ ether_phy_flow_control_t

| enum ether_phy_flow_control_t | |
|---|---|
| Flow control functionality | |
| Enumerator | |
| ETHER_PHY_FLOW_CONTROL_DISABLE | Disable flow control functionality. |
| ETHER_PHY_FLOW_CONTROL_ENABLE | Enable flow control functionality with pause frames. |

### ◆ ether_phy_link_speed_t

| enum ether_phy_link_speed_t | |
|---|---|
| Link speed | |
| Enumerator | |
| ETHER_PHY_LINK_SPEED_NO_LINK | Link is not established. |
| ETHER_PHY_LINK_SPEED_10H | Link status is 10Mbit/s and half duplex. |
| ETHER_PHY_LINK_SPEED_10F | Link status is 10Mbit/s and full duplex. |
| ETHER_PHY_LINK_SPEED_100H | Link status is 100Mbit/s and half duplex. |
| ETHER_PHY_LINK_SPEED_100F | Link status is 100Mbit/s and full duplex. |
| ETHER_PHY_LINK_SPEED_1000H | Link status is 1000Mbit/s and half duplex. |
| ETHER_PHY_LINK_SPEED_1000F | Link status is 1000Mbit/s and full duplex. |

### ◆ ether_phy_mii_type_t

| enum ether_phy_mii_type_t | |
|---|---|
| Media-independent interface | |
| Enumerator | |
| ETHER_PHY_MII_TYPE_MII | MII. |
| ETHER_PHY_MII_TYPE_RMII | RMII. |
| ETHER_PHY_MII_TYPE_GMII | GMII. |
| ETHER_PHY_MII_TYPE_RGMII | RGMII. |

### 5.3.11.8 Ethernet Switch Interface

Interfaces » Networking

#### Detailed Description

Interface for Ethernet Switch functions.

# Summary

The Ether Switch module provides an API for ethernet switch peripheral. And the general ethernet switch peripheral have forwarding functionality.

#### Data Structures

| | |
|---|---|
| struct | ether_switch_callback_args_t |
| struct | ether_switch_cfg_t |
| struct | ether_switch_api_t |
| struct | ether_switch_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | ether_switch_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | ether_switch_event_t |

## Data Structure Documentation

### ◆ ether_switch_callback_args_t

| struct ether_switch_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| uint32_t | channel | Device channel number. |
| uint32_t | ports | Bitmap of ports on which the interrupt occurred. |
| uint32_t | queue_index | Queue index where a interrupt occurs. |
| ether_switch_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. Set in ether_switch_api_t::open function in ether_switch_cfg_t. |

### ◆ ether_switch_cfg_t

| struct ether_switch_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel. |
| | |
| IRQn_Type | irq |
| | MCU interrupt number. |
| | |
| uint8_t | ipl |
| | MCU interrupt priority. |
| | |
| void(* | p_callback )(ether_switch_callback_args_t *p_args) |
| | Callback provided when an ISR occurs. |
| | |
| void * | p_context |

| | |
|---|---|
| void const * | p_extend |
| | Placeholder for user extension. |

## Field Documentation

### ◆ p_context

void* ether_switch_cfg_t::p_context

Placeholder for user data. Passed to the user callback in ether_switch_callback_args_t.

### ◆ ether_switch_api_t

struct ether_switch_api_t

Functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg) |
| fsp_err_t(* | close )(ether_switch_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* ether_switch_api_t::open) (ether_switch_ctrl_t *const p_ctrl, ether_switch_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

| fsp_err_t(* ether_switch_api_t::close) (ether_switch_ctrl_t *const p_ctrl) |
|---|

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ ether_switch_instance_t

| struct ether_switch_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| ether_switch_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ether_switch_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ether_switch_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ether_switch_ctrl_t

| typedef void ether_switch_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ ether_switch_event_t

| enum ether_switch_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| ETHER_SWITCH_EVENT_RX_COMPLETE | A descriptor complete to receive a flame. |
| ETHER_SWITCH_EVENT_TX_COMPLETE | A descriptor complete to transmit a flame. |
| ETHER_SWITCH_EVENT_RX_QUEUE_FULL | A RX descriptor queue is full. |
| ETHER_SWITCH_EVENT_RX_MESSAGE_LOST | Receive a frame when a RX descriptor queue is full. |
| ETHER_SWITCH_EVENT_TAS_ERROR | TAS gate error. |

## 5.3.11.9 GPTP Interface

Interfaces » Networking

### Detailed Description

Interface for gPTP timing.

# Summary

The gPTP API provides a generic interface for gPTP timer operation.

### Data Structures

| | |
|---|---|
| struct | gptp_timer_value_t |
| struct | gptp_timer_cfg_t |
| struct | gptp_callback_args_t |
| struct | gptp_cfg_t |
| struct | gptp_api_t |
| struct | gptp_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | gptp_ctrl_t |

## Data Structure Documentation

### ◆ gptp_timer_value_t

| struct gptp_timer_value_t | | |
|---|---|---|
| Timer value. | | |
| Data Fields | | |
| uint16_t | time_sec_upper | Second(Upper 16 bit). |
| uint32_t | time_sec_lower | Second(Lower 32 bit). |
| uint32_t | time_nsec | Nanosecond. |

### ◆ gptp_timer_cfg_t

| struct gptp_timer_cfg_t | | |
|---|---|---|
| Configuration of gPTP timer. | | |
| Data Fields | | |
| uint8_t | clock_period | Timer increment value. |

### ◆ gptp_callback_args_t

| struct gptp_callback_args_t | | |
|---|---|---|
| GPTP callback arguments definitions. | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in gptp_api_t::open function in gptp_cfg_t. |

### ◆ gptp_cfg_t

| struct gptp_cfg_t | |
|---|---|
| Configuration parameters. | |
| **Data Fields** | |
| void(* | p_callback )(gptp_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to the user callback in gptp_callback_args_t. |
| | |
| void const * | p_extend |

| | |
|---|---|
| | Placeholder for user extension. |
| | |

### ◆ gptp_api_t

| struct gptp_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(gptp_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | timerCfg )(gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg) |
| | |
| fsp_err_t(* | start )(gptp_ctrl_t *const p_ctrl, uint8_t timer) |
| | |
| fsp_err_t(* | stop )(gptp_ctrl_t *const p_ctrl, uint8_t timer) |
| | |
| fsp_err_t(* | timerValueGet )(gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value) |
| | |
| fsp_err_t(* | timerOffsetSet )(gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset) |
| | |
| fsp_err_t(* | timerRateSet )(gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate) |
| | |
| fsp_err_t(* | callbackSet )(gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, gptp_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* gptp_api_t::open) (gptp_ctrl_t *const p_ctrl, gptp_cfg_t const *const p_cfg)

Open driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* gptp_api_t::close) (gptp_ctrl_t *const p_ctrl)

Close driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ timerCfg

fsp_err_t(* gptp_api_t::timerCfg) (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_cfg_t const *const p_timer_cfg)

Configure gptp timer parameters.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |
| [in] | p_timer_cfg | Configuration of the timer. |

### ◆ start

fsp_err_t(* gptp_api_t::start) (gptp_ctrl_t *const p_ctrl, uint8_t timer)

Start gptp timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |

### ◆ stop

fsp_err_t(* gptp_api_t::stop) (gptp_ctrl_t *const p_ctrl, uint8_t timer)

Stop gptp timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |

### ◆ timerValueGet

fsp_err_t(* gptp_api_t::timerValueGet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, gptp_timer_value_t *const p_timer_value)

Get the current time value to gptp timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |
| [out] | p_timer_value | Pointer to timer value structure. |

### ◆ timerOffsetSet

fsp_err_t(* gptp_api_t::timerOffsetSet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, int64_t offset)

Set time offset correction to gptp timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |
| [in] | offset | Time offset value. |

### ◆ timerRateSet

fsp_err_t(* gptp_api_t::timerRateSet) (gptp_ctrl_t *const p_ctrl, uint8_t timer, uint32_t rate)

Set clock rate correction to gptp timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | timer | Timer index. |
| [in] | rate | Clock rate value. |

### ◆ callbackSet

| fsp_err_t(* gptp_api_t::callbackSet) (gptp_ctrl_t *const p_ctrl, void(*p_callback)(gptp_callback_args_t *), void *const p_context, gptp_callback_args_t *const p_callback_memory) |
|---|

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_callback | Callback function. |
| [in] | p_context | Pointer to send to callback function. |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ gptp_instance_t

| struct gptp_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| gptp_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| gptp_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| gptp_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ gptp_ctrl_t

| typedef void gptp_ctrl_t |
|---|
| Control block. Allocate an instance specific control block to pass into the API calls. |

## 5.3.11.10 PTP Interface

Interfaces » Networking

### Detailed Description

Interface for PTP functions.

# Summary

The PTP interface provides the functionality for using PTP.

### Data Structures

| | |
|---|---|
| struct | ptp_clock_properties_t |
| struct | ptp_time_t |
| struct | ptp_message_flags_t |
| struct | ptp_message_header_t |
| struct | ptp_message_sync_t |
| struct | ptp_message_pdelay_req_t |
| struct | ptp_message_pdelay_resp_t |
| struct | ptp_message_announce_t |
| struct | ptp_message_signaling_t |
| struct | ptp_message_management_t |
| struct | ptp_message_t |
| struct | ptp_callback_args_t |
| struct | ptp_pulse_timer_common_cfg_t |
| struct | ptp_pulse_timer_cfg_t |
| struct | ptp_sync_state_cfg_t |
| struct | ptp_synfp_cfg_t |
| struct | ptp_synfp_cfg_t.ether |
| struct | ptp_synfp_cfg_t.ipv4 |
| struct | ptp_stca_cfg_t |
| struct | ptp_cfg_t |

| | |
|---|---|
| struct | ptp_api_t |
| struct | ptp_instance_t |

### Typedefs

| | |
|---|---|
| typedef enum PTP_PACKED e_ptp_ctrl_field | ptp_ctrl_field_t |
| typedef ptp_message_sync_t | ptp_message_delay_req_t |
| typedef ptp_message_sync_t | ptp_message_follow_up_t |
| typedef ptp_message_pdelay_resp_t | ptp_message_delay_resp_t |
| typedef ptp_message_delay_resp_t | ptp_message_pdelay_resp_follow_up_t |

### Enumerations

| | |
|---|---|
| enum | ptp_message_type_t |
| enum | ptp_port_state_t |
| enum | ptp_clock_delay_mechanism_t |
| enum | ptp_frame_format_t |
| enum | ptp_frame_filter_mode_t |
| enum | ptp_stca_clock_freq_t |
| enum | ptp_stca_clock_sel_t |
| enum | ptp_message_interval_t |
| enum | ptp_clock_correction_mode_t |
| enum | ptp_event_t |
| enum | ptp_ethernet_phy_interface_t |

### Variables

| | |
|---|---|
| enum PTP_PACKED | e_ptp_ctrl_field |

## Data Structure Documentation

### ◆ ptp_clock_properties_t

| struct ptp_clock_properties_t | | |
|---|---|---|
| Clock properties used in the best master clock algorithm (BMCA) in order to determine the grandmaster clock.<br>In master mode, these properties will be advertised in announce messages.<br>Note: The final property used in BMCA is the clock ID. This is usually configured at runtime because it is often based on the hardware address. | | |
| Data Fields | | |
| uint8_t | priority1 | Priority1 value used in best master calculation. |
| uint8_t | cclass | Class value. |
| uint8_t | accuracy | Accuracy of the clock. |
| uint16_t | variance | Variance of the clock. |
| uint8_t | priority2 | Priority2 value used as secondary priority in best master calculation. |

### ◆ ptp_time_t

| struct ptp_time_t | | |
|---|---|---|
| Structure for storing time with nanosecond precision . | | |
| Data Fields | | |
| uint16_t | seconds_upper | Upper 16 bits of the seconds. |
| uint32_t | seconds_lower | Lower 32 bits of the seconds. |
| uint32_t | nanoseconds | Nanoseconds. |

### ◆ ptp_message_flags_t

| struct ptp_message_flags_t |
|---|
| Flags field in PTP message header. |

### ◆ ptp_message_header_t

| struct ptp_message_header_t | | |
|---|---|---|
| Commom PTP Message Header. | | |
| Data Fields | | |
| uint8_t | message_type: 4 | The message type. |
| uint8_t | sdoid_major: 4 | Standard Organization ID Major. |
| uint8_t | version: 4 | PTP Version. |

| | | |
|---|---|---|
| uint8_t | minor_version: 4 | PTP Minor Version. |
| uint16_t | message_length | The total message length (Including the header). |
| uint8_t | domain | The clock domain. |
| uint8_t | sdoid_minor: 8 | Standard Organization ID minor. |
| ptp_message_flags_t | flags | Flags set in the message. |
| uint64_t | correction_field | Correction Field that is updated when a message passes through a transparent clock. |
| uint32_t | reserved | |
| uint8_t | clock_id[8] | Clock ID that the message was sent from. |
| uint16_t | source_port_id | Port ID that the message was sent from. |
| uint16_t | sequence_id | Sequence ID of the message. |
| ptp_ctrl_field_t | control_field | Control field (Message specifc). |
| uint8_t | log_message_interval | Logbase2 of the message period. |

### ◆ ptp_message_sync_t

| struct ptp_message_sync_t | | |
|---|---|---|
| Sync Message Type (0x00). | | |
| Data Fields | | |
| ptp_time_t | origin_timestamp | Timestamp when the message was transmitted. |

### ◆ ptp_message_pdelay_req_t

| struct ptp_message_pdelay_req_t | | |
|---|---|---|
| PDelay_req Message Type (0x02). | | |
| Data Fields | | |
| ptp_time_t | origin_timestamp | Timestamp when the message was transmitted. |
| uint8_t | reserved[10] | |

### ◆ ptp_message_pdelay_resp_t

| struct ptp_message_pdelay_resp_t | | |
|---|---|---|
| PDelay_resp Message Type (0x03). | | |
| Data Fields | | |
| | | |

| | | |
|---|---|---|
| ptp_time_t | origin_timestamp | Timestamp when the message was transmitted. |
| uint8_t | source_port_identity[10] | Clock ID + sourcePortId. |

### ◆ ptp_message_announce_t

| struct ptp_message_announce_t | | |
|---|---|---|
| Announce Message Type (0x0B). | | |
| Data Fields | | |
| ptp_time_t | origin_timestamp | Timestamp when the message was transmitted. |
| uint16_t | current_utc_offset | Offset from UTC in seconds. |
| uint8_t | reserved | |
| ptp_clock_properties_t | clock_properties | Clock properties used in Best Master Clock Algorithm. |
| uint8_t | clock_id[8] | Clock ID that the message was sent from. |
| uint16_t | steps_removed | The number of boundary clocks between the clock and the grand master clock. |
| uint8_t | time_source | The source of time (Eg. INTERNAL_OSC). |

### ◆ ptp_message_signaling_t

| struct ptp_message_signaling_t | | |
|---|---|---|
| Signaling Message Type (0x0C). | | |
| Data Fields | | |
| uint8_t | target_clock_id[8] | ID of the target PTP instance. |
| uint16_t | target_port_id | Port of the target PTP instance. |

### ◆ ptp_message_management_t

| struct ptp_message_management_t | | |
|---|---|---|
| Management Message Type (0x0D). | | |
| Data Fields | | |
| uint8_t | target_clock_id[8] | ID of the target PTP instance. |
| uint16_t | target_port_id | Port of the target PTP instance. |
| uint8_t | starting_boundary_hops | The starting number of times the message is retransmitted by boundary clocks. |
| uint8_t | boundary_hops | The remaining number of retransmissions. |

| | | |
|---|---|---|
| uint8_t | action | The action that will be taken on reception of the message. |
| uint8_t | reserved | |

### ◆ ptp_message_t

| struct ptp_message_t | | |
|---|---|---|
| Complete PTP Message. | | |
| Data Fields | | |
| ptp_message_header_t | header | Header of the message. |
| union ptp_message_t | __unnamed__ | |

### ◆ ptp_callback_args_t

| struct ptp_callback_args_t | | |
|---|---|---|
| Arguments passed to p_ptp_callback. | | |
| Data Fields | | |
| ptp_event_t | event | Event that caused the callback. |
| ptp_message_t const * | p_message | The message received (PTP message fields will be little endian). |
| uint8_t const * | p_tlv_data | Start of TLV data (TLV data will be big endian). |
| uint16_t | tlv_data_size | Total bytes of TLV data. |
| uint32_t | pulse_timer_channel | Channel of the pulse timer that caused ptp_event_t::PTP_EVENT_PULSE_TIMER_MINT_RISING_EDGE. |
| void * | p_context | Context value set in the configuration. |

### ◆ ptp_pulse_timer_common_cfg_t

| struct ptp_pulse_timer_common_cfg_t | | |
|---|---|---|
| Structure for configuring the IPLS IRQ settings that are common to all pulse timer channels. | | |
| Data Fields | | |
| ptp_enable_t | ipls_rising_irq | Enable the IPLS IRQ when a rising edge is detected. |
| ptp_enable_t | ipls_falling_irq | Enable the IPLS IRQ when a falling edge is detected. |
| ptp_enable_t | ipls_rising_irq_auto_clear | Auto disable the rising edge IRQ after the first rising edge is detected. |
| | | |

| | | |
|---|---|---|
| ptp_enable_t | ipls_falling_irq_auto_clear | Auto disable the falling edge IRQ after the first falling edge is detected. |

### ◆ ptp_pulse_timer_cfg_t

| struct ptp_pulse_timer_cfg_t | | |
|---|---|---|
| Structure for configuring a pulse timer channel. | | |
| Data Fields | | |
| ptp_time_t | start_time | The exact time when the timer will start. |
| uint32_t | period | The period of the timer in nanoseconds. |
| uint32_t | pulse | The pulse width of the timer in nanoseconds. |
| ptp_enable_t | mint_rising_irq | Enable MINT rising edge IRQ. |
| ptp_enable_t | ipls_rising_event | Enable IPLS rising edge ELC event. |
| ptp_enable_t | ipls_falling_event | Enable IPLS falling edge ELC event. |
| ptp_enable_t | ipls_rising_event_auto_clear | Enable IPLS rising edge ELC event. |
| ptp_enable_t | ipls_falling_event_auto_clear | Enable IPLS falling edge ELC event. |
| ptp_enable_t | ipls_irq_source | Enable using this channel as a source for the IPLS IRQ. |

### ◆ ptp_sync_state_cfg_t

| struct ptp_sync_state_cfg_t | | |
|---|---|---|
| Configuration settings for determining when the PTP clock is synchronized. | | |
| Data Fields | | |
| uint64_t | threshold | The maximum clock offset required to transition between synchronization states. |
| uint8_t | count | The number of times the clock must be above the threshold in order to transition between synchronization states. |

### ◆ ptp_synfp_cfg_t

| struct ptp_synfp_cfg_t |
|---|
| Configuration settings for the SYNFP. |
| Data Fields |

| | | |
|---|---|---|
| ptp_ethernet_phy_interface_t | ethernet_phy_interface | The type of interface used to communicate with the PHY. |
| ptp_frame_format_t | frame_format | Frame format used to transport PTP messages. |
| ptp_frame_filter_mode_t | frame_filter | Frame filter mode. |
| uint8_t | clock_domain | Clock domain that the clock operates in. |
| ptp_enable_t | clock_domain_filter | Filter out messages from other clock domains. |
| ptp_message_interval_t | announce_interval | Interval for transmitting announce messages. |
| ptp_message_interval_t | sync_interval | Interval for transmitting sync messages. |
| ptp_message_interval_t | delay_req_interval | Interval for transmitting delay_req messages. |
| uint32_t | message_timeout | Timeout in milliseconds for receiving PTP messages. |
| ptp_clock_properties_t | clock_properties | Clock properties used in annonce messages. |
| uint8_t | timesource | TimeSource field used in announce messages. |
| uint8_t * | p_multicast_addr_filter | Filter for multicast packets. |
| struct ptp_synfp_cfg_t | ether | Valid if frame_format is set to Ethernet II or IEEE 802.3. |
| struct ptp_synfp_cfg_t | ipv4 | Valid if frame_format is set to IPV4_UDP. |

### ◆ ptp_synfp_cfg_t.ether

| struct ptp_synfp_cfg_t.ether | | |
|---|---|---|
| Valid if frame_format is set to Ethernet II or IEEE 802.3. | | |
| Data Fields | | |
| uint8_t * | p_primary_mac_addr | The MAC address to send primary messages. |
| uint8_t * | p_pdelay_mac_addr | The MAC address to send p2p messages. |

### ◆ ptp_synfp_cfg_t.ipv4

| struct ptp_synfp_cfg_t.ipv4 | | |
|---|---|---|
| Valid if frame_format is set to IPV4_UDP. | | |
| Data Fields | | |
| uint32_t | primary_ip_addr | The IP address to send primary |

| | | |
|---|---|---|
| | | messages. |
| uint32_t | pdelay_ip_addr | The IP address to send pdelay messages. |
| uint8_t | event_tos | Type of service for event messages. |
| uint8_t | general_tos | Type of service for general messages. |
| uint8_t | primary_ttl | Time to live for primary messages. |
| uint8_t | pdelay_ttl | Time to live for pdelay messages. |
| uint16_t | event_udp_port | The port to send event messages. |
| uint16_t | general_udp_port | The port to send general messages. |

### ◆ ptp_stca_cfg_t

| struct ptp_stca_cfg_t | | |
|---|---|---|
| Configuration settings for the STCA. | | |
| Data Fields | | |
| ptp_stca_clock_freq_t | clock_freq | Select the clock frequency of the STCA. |
| ptp_stca_clock_sel_t | clock_sel | Select the input clock to the STCA. |
| ptp_clock_correction_mode_t | clock_correction_mode | Select the clock correction mode. |
| uint8_t | gradient_worst10_interval | Select the interval for the gradient worst10 acquisition. |
| ptp_sync_state_cfg_t | sync_threshold | Configure the synchronization threshold. |
| ptp_sync_state_cfg_t | sync_loss_threshold | Configure the SYnchronization lost threshold. |

### ◆ ptp_cfg_t

| struct ptp_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| ptp_synfp_cfg_t | synfp |
| | Configuration settings for the SYNFP. |
| | |

| | |
|---|---|
| ptp_stca_cfg_t | stca |
| | Configuration settings for the STCA. |
| | |
| edmac_instance_t * | p_edmac_instance |
| | Pointer to PTP edmac instance. |
| | |
| uint16_t | buffer_size |
| | The maximum Ethernet packet size that can be transmitted or received. |
| | |
| uint8_t ** | p_rx_buffers |
| | Pointer to list of buffers used to receive PTP packets. |
| | |
| uint8_t ** | p_tx_buffers |
| | Pointer to list of buffers used to transmit PTP packets. |
| | |
| IRQn_Type | mint_irq |
| | Interrupt number for PTP event IRQ. |
| | |
| IRQn_Type | ipls_irq |
| | Interrupt number for PTP timer IRQ. |
| | |
| uint8_t | mint_ipl |
| | Interrupt priority of the PTP event IRQ. |
| | |
| uint8_t | ipls_ipl |
| | Interrupt priority of the PTP timer IRQ. |
| | |
| | |

| | |
|---|---|
| void(* | p_callback )(ptp_callback_args_t *p_args) |
| | |

## Field Documentation

### ◆ p_callback

| |
|---|
| void(* ptp_cfg_t::p_callback) (ptp_callback_args_t *p_args) |
| Callback for handling received PTP events. |

### ◆ ptp_api_t

| |
|---|
| struct ptp_api_t |
| Timer API structure. General timer functions implemented at the HAL layer follow this API. |
| **Data Fields** |

| | |
|---|---|
| fsp_err_t(* | open )(ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | macAddrSet )(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr) |
| | |
| fsp_err_t(* | ipAddrSet )(ptp_ctrl_t *const p_ctrl, uint32_t ip_addr) |
| | |
| fsp_err_t(* | localClockIdSet )(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id) |
| | |
| fsp_err_t(* | masterClockIdSet )(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id) |
| | |
| fsp_err_t(* | messageFlagsSet )(ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags) |
| | |
| fsp_err_t(* | currentUtcOffsetSet )(ptp_ctrl_t *const p_ctrl, uint16_t offset) |
| | |
| fsp_err_t(* | portStateSet )(ptp_ctrl_t *const p_ctrl, uint32_t state) |
| | |
| fsp_err_t(* | messageSend )(ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size) |
| | |
| fsp_err_t(* | localClockValueSet )(ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time) |

| | |
|---|---|
| | |
| fsp_err_t(* | localClockValueGet )(ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time) |
| | |
| fsp_err_t(* | pulseTimerCommonConfig )(ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *p_timer_cfg) |
| | |
| fsp_err_t(* | pulseTimerEnable )(ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg) |
| | |
| fsp_err_t(* | pulseTimerDisable )(ptp_ctrl_t *const p_ctrl, uint32_t channel) |
| | |
| fsp_err_t(* | close )(ptp_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* ptp_api_t::open) (ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)

Initial configuration.

***Note***

*To reconfigure after calling this function, call ptp_api_t::close first.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ macAddrSet

fsp_err_t(* ptp_api_t::macAddrSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)

Set the MAC address for the PTP.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_hw_addr | Pointer to the 6 byte MAC address. |

### ◆ ipAddrSet

fsp_err_t(* ptp_api_t::ipAddrSet) (ptp_ctrl_t *const p_ctrl, uint32_t ip_addr)

Set the IP address for the PTP.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | ip_addr | 32 bit IPv4 address of the PTP. |

### ◆ localClockIdSet

fsp_err_t(* ptp_api_t::localClockIdSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id)

Set the local clock ID (Usually based off of the PTP MAC address).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_clock_id | Pointer to 8 byte clock ID. |

### ◆ masterClockIdSet

fsp_err_t(* ptp_api_t::masterClockIdSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id)

Set the master clock ID (Usually obtained from previously received announce message).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_clock_id | Pointer to 8 byte clock ID. |
| [in] | port_id | The port on the master clock. |

### ◆ messageFlagsSet

fsp_err_t(* ptp_api_t::messageFlagsSet) (ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags)

Set the flags field for the given message type.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | message_type | The message type. |
| [in] | flags | Flags to set. |

### ◆ currentUtcOffsetSet

fsp_err_t(* ptp_api_t::currentUtcOffsetSet) (ptp_ctrl_t *const p_ctrl, uint16_t offset)

Sets the offsetFromMaster field in announce messages.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | offset | New currentUtcOffset value. |

### ◆ portStateSet

fsp_err_t(* ptp_api_t::portStateSet) (ptp_ctrl_t *const p_ctrl, uint32_t state)

Transition to a new clock state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | state | The state to transition into. |

### ◆ messageSend

fsp_err_t(* ptp_api_t::messageSend) (ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size)

Send a PTP message. Appropriate fields in the PTP message will be endian swapped. The application must ensure that the TLV data is in big endian format.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_message | Pointer to a PTP message. |
| [in] | p_tlv_data | Pointer to TLV data that is appended to the end of the PTP message. |
| [in] | tlv_data_size | Size of the TLV data in bytes. |

### ◆ localClockValueSet

fsp_err_t(* ptp_api_t::localClockValueSet) (ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time)

Set the local clock value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_time | Pointer to the new time setting. |

### ◆ localClockValueGet

fsp_err_t(* ptp_api_t::localClockValueGet) (ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time)

Get the local clock value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_time | Pointer to store the current time setting. |

### ◆ pulseTimerCommonConfig

fsp_err_t(* ptp_api_t::pulseTimerCommonConfig) (ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *p_timer_cfg)

Configuration that is common to all of the pulse timers.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_timer_cfg | Pointer to the pulse timer common configuration. |

### ◆ pulseTimerEnable

fsp_err_t(* ptp_api_t::pulseTimerEnable) (ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg)

Setup a pulse timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | channel | The pulse timer channel to setup. |
| [in] | p_timer_cfg | Pointer to the pulse timer configuration. |

### ◆ pulseTimerDisable

fsp_err_t(* ptp_api_t::pulseTimerDisable) (ptp_ctrl_t *const p_ctrl, uint32_t channel)

Stop a pulse timer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | channel | The pulse timer channel to stop. |

### ◆ close

fsp_err_t(* ptp_api_t::close) (ptp_ctrl_t *const p_ctrl)

Stop PTP operation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ ptp_instance_t

struct ptp_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| ptp_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ptp_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ptp_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ ptp_ctrl_field_t

| typedef enum PTP_PACKED e_ptp_ctrl_field ptp_ctrl_field_t |
| --- |
| The control field for PTP message header. |

### ◆ ptp_message_delay_req_t

| typedef ptp_message_sync_t ptp_message_delay_req_t |
| --- |
| Delay_req Message Type (0x01). |

### ◆ ptp_message_follow_up_t

| typedef ptp_message_sync_t ptp_message_follow_up_t |
| --- |
| Follow_up Message Type (0x08). |

### ◆ ptp_message_delay_resp_t

| typedef ptp_message_pdelay_resp_t ptp_message_delay_resp_t |
| --- |
| Delay_resp Message Type (0x09). |

### ◆ ptp_message_pdelay_resp_follow_up_t

| typedef ptp_message_delay_resp_t ptp_message_pdelay_resp_follow_up_t |
| --- |
| PDelay_resp_follow_up Message Type (0x0A). |

## Enumeration Type Documentation

### ◆ ptp_message_type_t

| enum ptp_message_type_t | |
|---|---|
| Standard PTP message types. | |
| Enumerator | |
| PTP_MESSAGE_TYPE_SYNC | Sync Message Type. |
| PTP_MESSAGE_TYPE_DELAY_REQ | Delay_req Message Type. |
| PTP_MESSAGE_TYPE_PDELAY_REQ | PDelay_req Message Type. |
| PTP_MESSAGE_TYPE_PDELAY_RESP | PDelay_resp Message Type. |
| PTP_MESSAGE_TYPE_FOLLOW_UP | Follow_up Message Type. |
| PTP_MESSAGE_TYPE_DELAY_RESP | Delay_resp Message Type. |
| PTP_MESSAGE_TYPE_PDELAY_RESP_FOLLOW_UP | PDelay_resp_follow_up Message Type. |
| PTP_MESSAGE_TYPE_ANNOUNCE | Announce Message Type. |
| PTP_MESSAGE_TYPE_SIGNALING | Signaling Message Type. |
| PTP_MESSAGE_TYPE_MANAGEMENT | Management Message Type. |

### ◆ ptp_port_state_t

| enum ptp_port_state_t | |
|---|---|
| Possible states that the PTP instance can be in. | |
| Enumerator | |
| PTP_PORT_STATE_GENERATE_ANNOUNCE | Generate Announce Messages. |
| PTP_PORT_STATE_GENERATE_SYNC | Generate Sync Messages. |
| PTP_PORT_STATE_GENERATE_DELAY_REQ | Generate Delay_req Messages. |
| PTP_PORT_STATE_GENERATE_PDELAY_REQ | Generate PDelay_req Messages. |
| PTP_PORT_STATE_RECEIVE_ANNOUNCE | Receive Announce Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_SYNC | Receive Sync Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_FOLLOW_UP | Receive Follow_up Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_DELAY_REQ | Receive Delay_req Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_DELAY_RESP | Receive Delay_resp Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_PDELAY_REQ | Receive PDelay_req Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_PDELAY_RESP | Receive PDelay_resp Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_PDELAY_RESP_FOLLOW_UP | Receive PDelay_resp_follow_up Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_MANAGEMENT | Receive Management Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_RECEIVE_SIGNALING | Receive Signaling Messages from ptp_cfg_t::p_callback. |
| PTP_PORT_STATE_PROCESS_SYNC | Enable Sync Message processing. |
| PTP_PORT_STATE_PROCESS_FOLLOW_UP | Enable Follow_up Message processing. |
| | |

| | |
|---|---|
| PTP_PORT_STATE_PROCESS_DELAY_REQ | Enable Delay_req Message processing. |
| PTP_PORT_STATE_PROCESS_DELAY_RESP | Enable Delay_resp Message processing. |
| PTP_PORT_STATE_PROCESS_PDELAY_REQ | Enable PDelay_req Message processing. |
| PTP_PORT_STATE_PROCESS_PDELAY_RESP | Enable PDelay_resp Message processing. |
| PTP_PORT_STATE_PROCESS_PDELAY_RESP_FOLLOW_UP | Enable PDelay_resp_follow_up Message processing. |
| PTP_PORT_STATE_PASSIVE | Configure the PTP instance to only receive Announce, Management, and Signaling Messages. |
| PTP_PORT_STATE_E2E_MASTER | Configure the PTP instance to operate as a E2E Master. |
| PTP_PORT_STATE_E2E_SLAVE | Configure the PTP instance to operate as a E2E Slave. |
| PTP_PORT_STATE_P2P_MASTER | Configure the PTP instance to operate as a P2P Master. |
| PTP_PORT_STATE_P2P_SLAVE | Configure the PTP instance to operate as a P2P Slave. |
| PTP_PORT_STATE_RECEIVE_ALL | Configure the PTP instance to receive all messages. |
| PTP_PORT_STATE_DISABLE | Disable all PTP message generation, processing, and reception. |

### ◆ ptp_clock_delay_mechanism_t

| enum ptp_clock_delay_mechanism_t | |
|---|---|
| The mechanism used for delay messages. | |
| Enumerator | |
| PTP_CLOCK_DELAY_MECHANISM_E2E | End to end delay mechanism. |
| PTP_CLOCK_DELAY_MECHANISM_P2P | Peer to peer delay mechanism. |

### ◆ ptp_frame_format_t

| enum ptp_frame_format_t | |
|---|---|
| Frame formats that PTP messages can be encapsulated in. | |
| Enumerator | |
| PTP_FRAME_FORMAT_ETHERII | Send PTP messages using Ethernet II frames. |
| PTP_FRAME_FORMAT_IEEE802_3 | Send PTP messages using IEEE802_3 frames. |
| PTP_FRAME_FORMAT_ETHERII_IPV4_UDP | Send PTP messages using Ethernet II frames with an IP and UDP header. |
| PTP_FRAME_FORMAT_IEEE802_3_IPV4_UDP | Send PTP messages using IEEE802.3 frames with an IP and UDP header. |

### ◆ ptp_frame_filter_mode_t

| enum ptp_frame_filter_mode_t | |
|---|---|
| Filter PTP messages based on destination MAC address. Messages that pass the filter will be transferred to the ETHERC EDMAC. | |
| Enumerator | |
| PTP_FRAME_FILTER_MODE_EXT_PROMISCUOUS_MODE | Receive all packets. |
| PTP_FRAME_FILTER_MODE_UNICAST_MULTICAST | Receive all unicast packets destined for the PTP and all multicast packets. |
| PTP_FRAME_FILTER_MODE_UNICAST_MULTICAST_FILTERED | Receive Unicast packets destined for the PTP and filter configured multicast packets. |
| PTP_FRAME_FILTER_MODE_UNICAST | Receive unicast packets destined for the PTP. |

### ◆ ptp_stca_clock_freq_t

| enum ptp_stca_clock_freq_t | |
|---|---|
| STCA input clock frequency. | |
| Enumerator | |
| PTP_STCA_CLOCK_FREQ_20MHZ | 20 Mhz Input Clock |
| PTP_STCA_CLOCK_FREQ_25MHZ | 25 Mhz Input Clock |
| PTP_STCA_CLOCK_FREQ_50MHZ | 50 Mhz Input Clock |
| PTP_STCA_CLOCK_FREQ_100MHZ | 100 Mhz Input Clock |

### ◆ ptp_stca_clock_sel_t

| enum ptp_stca_clock_sel_t | |
|---|---|
| STCA input clock selection. | |
| Enumerator | |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_1 | PCLKA. |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_2 | PCLKA Divided by 2. |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_3 | PCLKA Divided by 3. |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_4 | PCLKA Divided by 4. |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_5 | PCLKA Divided by 5. |
| PTP_STCA_CLOCK_SEL_PCLKA_DIV_6 | PCLKA Divided by 6. |
| PTP_STCA_CLOCK_SEL_REF50CK0 | 50-MHz Reference Signal for timing in RMII mode (STCA clock frequency is 25 Mhz when REF50CK0 is used). |

### ◆ ptp_message_interval_t

| enum ptp_message_interval_t | |
|---|---|
| Message interval for transmitting PTP messages. | |
| Enumerator | |
| PTP_MESSAGE_INTERVAL_1_128 | 1 / 128 seconds |
| PTP_MESSAGE_INTERVAL_1_64 | 1 / 64 seconds |
| PTP_MESSAGE_INTERVAL_1_32 | 1 / 32 seconds |
| PTP_MESSAGE_INTERVAL_1_16 | 1 / 16 seconds |
| PTP_MESSAGE_INTERVAL_1_8 | 1 / 8 seconds |
| PTP_MESSAGE_INTERVAL_1_4 | 1 / 4 seconds |
| PTP_MESSAGE_INTERVAL_1_2 | 1 / 2 seconds |
| PTP_MESSAGE_INTERVAL_1 | 1 seconds |
| PTP_MESSAGE_INTERVAL_2 | 2 seconds |
| PTP_MESSAGE_INTERVAL_4 | 4 seconds |
| PTP_MESSAGE_INTERVAL_8 | 8 seconds |
| PTP_MESSAGE_INTERVAL_16 | 16 seconds |
| PTP_MESSAGE_INTERVAL_32 | 32 seconds |
| PTP_MESSAGE_INTERVAL_64 | 64 seconds |

### ◆ ptp_clock_correction_mode_t

| enum ptp_clock_correction_mode_t | |
|---|---|
| Clock correction mode. | |
| Enumerator | |
| PTP_CLOCK_CORRECTION_MODE1 | Correct the local clock using the current offsetFromMaster value. |
| PTP_CLOCK_CORRECTION_MODE2 | Correct the local clock using the calculated clock gradient. |

### ◆ ptp_event_t

| enum ptp_event_t | |
|---|---|
| PTP events provided by ptp_cfg_t::p_callback. | |
| Enumerator | |
| PTP_EVENT_SYNC_ACQUIRED | The local clock is synchronized to the master clock. |
| PTP_EVENT_SYNC_LOST | The local clock is not synchronized to the master clcok. |
| PTP_EVENT_SYNC_MESSAGE_TIMEOUT | A sync message has not been received for the configured time. |
| PTP_EVENT_WORST10_ACQUIRED | Gradient worst10 values has been calculated. |
| PTP_EVENT_OFFSET_FROM_MASTER_UPDATED | The offset from the master clock has been updated. |
| PTP_EVENT_LOG_MESSAGE_INT_CHANGED | The message interval was changed. |
| PTP_EVENT_MEAN_PATH_DELAY_UPDATED | The mean path delay has been updated. |
| PTP_EVENT_DELAY_RESP_TIMEOUT | A delay_resp has not been received for the configured time. |
| PTP_EVENT_LOG_MESSAGE_INT_OUT_OF_RANGE | The updated message interval is out of range. |
| PTP_EVENT_DELAY_REQ_FIFO_OVERFLOW | The FIFO buffer for storing information from received Delay_Req messages holds 32 or more entries. |
| PTP_EVENT_LOOP_RECEPTION_DETECTED | A packet with the same sourcePortIdendity as the local clock was received. |
| PTP_EVENT_CTRL_INFO_ABNORMALITY | A malformed frame was received (EDMAC, ETHERC, and EPTPC must be reset). |
| PTP_EVENT_DELAY_RESP_PROCESSING_HALTED | Processing of delay_resp messages has been halted. |
| PTP_EVENT_MESSAGE_GENERATION_HALTED | Generation of messages has been halted. |
| PTP_EVENT_MESSAGE_RECEIVED | A PTP message was received from the EDMAC. |
| PTP_EVENT_MESSAGE_TRANSMIT_COMPLETE | A PTP message has been transmitted. |
| | |

| | |
|---|---|
| PTP_EVENT_PULSE_TIMER_MINT_RISING_EDGE | A rising edge occurred on a pulse timer channel. |
| PTP_EVENT_PULSE_TIMER_IPLS_COMMON | A rising or falling edge occurred on any pulse timer channel. |

### ◆ ptp_ethernet_phy_interface_t

| enum ptp_ethernet_phy_interface_t | |
|---|---|
| The Ethernet PHY interface type. | |
| Enumerator | |
| PTP_ETHERNET_PHY_INTERFACE_MII | Media-independent interface. |
| PTP_ETHERNET_PHY_INTERFACE_RMII | Reduced media-independent interface. |

## Variable Documentation

### ◆ e_ptp_ctrl_field

| enum PTP_PACKED e_ptp_ctrl_field |
|---|
| The control field for PTP message header. |

## 5.3.11.11 WiFi Interface

Interfaces » Networking

### Functions

| | |
|---|---|
| WIFIReturnCode_t | WIFI_On (void)<br>Turns on Wi-Fi. More... |
| WIFIReturnCode_t | WIFI_Off (void)<br>Turns off Wi-Fi. More... |
| WIFIReturnCode_t | WIFI_ConnectAP (const WIFINetworkParams_t *const pxNetworkParams)<br>Connects to the Wi-Fi Access Point (AP) specified in the input. More... |
| WIFIReturnCode_t | WIFI_Disconnect (void) |

| | |
|---|---|
| | Disconnects from the currently connected Access Point. More... |
| WIFIReturnCode_t | WIFI_Reset (void)<br>Resets the Wi-Fi Module. More... |
| WIFIReturnCode_t | WIFI_SetMode (WIFIDeviceMode_t xDeviceMode)<br>Sets the Wi-Fi mode. More... |
| WIFIReturnCode_t | WIFI_GetMode (WIFIDeviceMode_t *pxDeviceMode)<br>Gets the Wi-Fi mode. More... |
| WIFIReturnCode_t | WIFI_NetworkAdd (const WIFINetworkProfile_t *const pxNetworkProfile, uint16_t *pusIndex)<br>Add a Wi-Fi Network profile. More... |
| WIFIReturnCode_t | WIFI_NetworkGet (WIFINetworkProfile_t *pxNetworkProfile, uint16_t usIndex)<br>Get a Wi-Fi network profile. More... |
| WIFIReturnCode_t | WIFI_NetworkDelete (uint16_t usIndex)<br>Delete a Wi-Fi Network profile. More... |
| WIFIReturnCode_t | WIFI_Ping (uint8_t *pucIPAddr, uint16_t usCount, uint32_t ulIntervalMS)<br>Ping an IP address in the network. More... |
| WIFIReturnCode_t | WIFI_GetIPInfo (WIFIIPConfiguration_t *pxIPInfo)<br>Get IP configuration (IP address, NetworkMask, Gateway and DNS server addresses). More... |
| WIFIReturnCode_t | WIFI_GetMAC (uint8_t *pucMac)<br>Retrieves the Wi-Fi interface's MAC address. More... |
| WIFIReturnCode_t | WIFI_GetHostIP (char *pcHost, uint8_t *pucIPAddr)<br>Retrieves the host IP address from a host name using DNS. More... |

| | |
|---|---|
| WIFIReturnCode_t | WIFI_Scan (WIFIScanResult_t *pxBuffer, uint8_t ucNumNetworks)<br>Perform a Wi-Fi network Scan. More... |
| WIFIReturnCode_t | WIFI_StartAP (void)<br>Start SoftAP mode. More... |
| WIFIReturnCode_t | WIFI_StopAP (void)<br>Stop SoftAP mode. More... |
| WIFIReturnCode_t | WIFI_ConfigureAP (const WIFINetworkParams_t *const pxNetworkParams)<br>Configure SoftAP. More... |
| WIFIReturnCode_t | WIFI_SetPMMode (WIFIPMMode_t xPMModeType, const void *pvOptionValue)<br>Set the Wi-Fi power management mode. More... |
| WIFIReturnCode_t | WIFI_GetPMMode (WIFIPMMode_t *pxPMModeType, void *pvOptionValue)<br>Get the Wi-Fi power management mode. More... |
| WIFIReturnCode_t | WIFI_RegisterEvent (WIFIEventType_t xEventType, WIFIEventHandler_t xHandler)<br>Register a Wi-Fi event Handler. More... |
| WIFIReturnCode_t | WIFI_IsConnected (const WIFINetworkParams_t *pxNetworkParams)<br>Check if the Wi-Fi is connected and the AP configuration matches the query. More... |
| WIFIReturnCode_t | WIFI_StartScan (WIFIScanConfig_t *pxScanConfig)<br>Start a Wi-Fi scan. More... |
| WIFIReturnCode_t | WIFI_GetScanResults (const WIFIScanResult_t **pxBuffer, uint16_t *ucNumNetworks)<br>Get Wi-Fi scan results. It should be called only after scan is completed. Scan results are sorted in decreasing rssi order. More... |

| | |
|---|---|
| WIFIReturnCode_t | WIFI_StartConnectAP (const WIFINetworkParams_t *pxNetworkParams)<br>Connect to the Wi-Fi Access Point (AP) specified in the input. More... |
| WIFIReturnCode_t | WIFI_StartDisconnect (void)<br>Wi-Fi station disconnects from AP. More... |
| WIFIReturnCode_t | WIFI_GetConnectionInfo (WIFIConnectionInfo_t *pxConnectionInfo)<br>Get Wi-Fi info of the connected AP. More... |
| WIFIReturnCode_t | WIFI_GetRSSI (int8_t *pcRSSI)<br>Get the RSSI of the connected AP. More... |
| WIFIReturnCode_t | WIFI_GetStationList (WIFIStationInfo_t *pxStationList, uint8_t *pcStationListSize)<br>SoftAP mode get connected station list. More... |
| WIFIReturnCode_t | WIFI_StartDisconnectStation (uint8_t *pucMac)<br>AP mode disconnecting a station. More... |
| WIFIReturnCode_t | WIFI_SetSpoofingMAC (uint8_t *pucMac)<br>Set Wi-Fi MAC addresses. More... |
| WIFIReturnCode_t | WIFI_SetCountryCode (const char *pcCountryCode)<br>Set country based configuration (including channel list, power table) More... |
| WIFIReturnCode_t | WIFI_GetCountryCode (char *pcCountryCode)<br>Get the currently configured country code. More... |
| WIFIReturnCode_t | WIFI_GetStatistic (WIFIStatisticInfo_t *pxStats)<br>Get the Wi-Fi statistics. More... |
| WIFIReturnCode_t | WIFI_GetCapability (WIFICapabilityInfo_t *pxCaps) |

Get the Wi-Fi capability. More...

## Detailed Description

Interface for common WiFi APIs.

*Note*

*This API has been moved over from the deprecated AWS iot_wifi API. It may not fully conform to FSP standards.*

### Data Structures

| | |
|---|---|
| struct | WIFIWEPKey_t |
| struct | WIFIWPAPassphrase_t |
| struct | WIFINetworkParams_t |
| struct | WIFIScanConfig_t |
| struct | WIFIScanResult_t |
| struct | WIFIStationInfo_t |
| struct | WIFINetworkProfile_t |
| struct | WIFIIPAddress_t |
| struct | WIFIIPConfiguration_t |
| struct | WIFIConnectionInfo_t |
| struct | WiFiEventInfoReady_t |
| struct | WiFiEventInfoScanDone_t |
| struct | WiFiEventInfoConnected_t |
| struct | WiFiEventInfoDisconnected_t |
| struct | WiFiEventInfoConnectionFailed_t |
| struct | WiFiEventInfoIPReady_t |
| struct | WiFiEventInfoAPStateChanged_t |
| struct | WiFiEventInfoAPStationConnected_t |
| struct | WiFiEventInfoAPStationDisconnected_t |

| | |
|---|---|
| struct | WiFiEventInfoRxDone_t |
| struct | WiFiEventInfoTxDone_t |
| struct | WIFIEvent_t |
| struct | WIFIStatisticInfo_t |
| struct | WIFICapabilityInfo_t |

### Macros

| | |
|---|---|
| #define | WIFI_WPS_SUPPORTED<br>Wi-Fi lower level supported feature mask. More... |

### Typedefs

| | |
|---|---|
| typedef void(* | WIFIEventHandler_t) (WIFIEvent_t *xEvent)<br>Wi-Fi event handler definition. More... |

### Enumerations

| | |
|---|---|
| enum | WIFIReturnCode_t |
| enum | WIFISecurity_t |
| enum | WIFIDeviceMode_t |
| enum | WIFIPMMode_t |
| enum | WIFIIPAddressType_t |
| enum | WIFIReason_t |
| enum | WIFIEventType_t |
| enum | WIFIBand_t |
| enum | WIFIPhyMode_t |
| enum | WIFIBandwidth_t |

## Data Structure Documentation

### ◆ WIFIWEPKey_t

struct WIFIWEPKey_t

| Wi-Fi WEP keys (64- and 128-bit keys only) | | |
|---|---|---|
| Data Fields | | |
| char | cKey[wificonfigMAX_WEPKEY_LEN] | WEP key (binary array, not C-string) |
| uint8_t | ucLength | Key length. |

### ◆ WIFIWPAPassphrase_t

| struct WIFIWPAPassphrase_t | | |
|---|---|---|
| Wi-Fi WPA/WPA2 passphrase | | |
| Data Fields | | |
| char | cPassphrase[wificonfigMAX_PASSPHRASE_LEN] | WPA passphrase (binary array, not C-string) |
| uint8_t | ucLength | Passphrase length (must be between 8 and 64 inclusive) |

### ◆ WIFINetworkParams_t

| struct WIFINetworkParams_t | | |
|---|---|---|
| Parameters passed to the WIFI_ConnectAP API for connection | | |
| Data Fields | | |
| uint8_t | ucSSID[wificonfigMAX_SSID_LEN] | SSID of the Wi-Fi network (binary array, not C-string) |
| uint8_t | ucSSIDLength | SSID length. |
| WIFISecurity_t | xSecurity | Wi-Fi Security. |
| union WIFINetworkParams_t | xPassword | |
| uint8_t | ucDefaultWEPKeyIndex | Default WEP key index. |
| uint8_t | ucChannel | Channel number. |

### ◆ WIFIScanConfig_t

| struct WIFIScanConfig_t | | |
|---|---|---|
| Wi-Fi scan configuration | | |
| Data Fields | | |
| uint8_t | ucSSID[wificonfigMAX_SSID_LEN] | SSID for targeted scan (binary array, not C-string) |
| uint8_t | ucSSIDLength | SSID length, 0 if broadcast scan. |
| uint8_t | ucChannel | Channel to scan (0 means all channels) |

### ◆ WIFIScanResult_t

| struct WIFIScanResult_t | | |
|---|---|---|
| Wi-Fi scan results | | |
| Data Fields | | |
| uint8_t | ucSSID[wificonfigMAX_SSID_LEN] | SSID of the Wi-Fi network (binary array, not C-string) |
| uint8_t | ucSSIDLength | SSID length. |
| uint8_t | ucBSSID[wificonfigMAX_BSSID_LEN] | BSSID of the Wi-Fi network (binary array, not C-string) |
| WIFISecurity_t | xSecurity | Security type of the Wi-Fi network. |
| int8_t | cRSSI | Signal strength of the Wi-Fi network. |
| uint8_t | ucChannel | Channel of the Wi-Fi network. |

### ◆ WIFIStationInfo_t

| struct WIFIStationInfo_t | | |
|---|---|---|
| Wi-Fi SoftAP connected station info | | |
| Data Fields | | |
| uint8_t | ucMAC[wificonfigMAX_BSSID_LEN] | MAC address of Wi-Fi station. |

### ◆ WIFINetworkProfile_t

| struct WIFINetworkProfile_t | | |
|---|---|---|
| Wi-Fi network parameters passed to the WIFI_NetworkAdd API | | |
| Data Fields | | |
| uint8_t | ucSSID[wificonfigMAX_SSID_LEN] | SSID of the Wi-Fi network to join with a NULL termination. |
| uint8_t | ucSSIDLength | SSID length not including NULL termination. |
| uint8_t | ucBSSID[wificonfigMAX_BSSID_LEN] | BSSID of the Wi-Fi network. |
| char | cPassword[wificonfigMAX_PASSPHRASE_LEN] | Password needed to join the AP. |
| uint8_t | ucPasswordLength | Password length not including null termination. |
| WIFISecurity_t | xSecurity | Wi-Fi Security.<br><br>See also<br>WIFISecurity_t |

### ◆ WIFIIPAddress_t

| struct WIFIIPAddress_t | | |
|---|---|---|
| Wi-Fi station IP address format | | |
| Data Fields | | |
| WIFIIPAddressType_t | xType | IP address type (only eWiFiIPAddressTypeV4 is currently supported) |
| uint32_t | ulAddress[IPV6_LENGTH] | IP address in binary form, use inet_ntop/inet_pton for conversion. |

### ◆ WIFIIPConfiguration_t

| struct WIFIIPConfiguration_t | | |
|---|---|---|
| IP address configuration | | |
| Data Fields | | |
| WIFIIPAddress_t | xIPAddress | IP address. |
| WIFIIPAddress_t | xNetMask | Network mask. |
| WIFIIPAddress_t | xGateway | Gateway IP address. |
| WIFIIPAddress_t | xDns1 | First DNS server IP address. |
| WIFIIPAddress_t | xDns2 | Second DNS server IP address. |

### ◆ WIFIConnectionInfo_t

| struct WIFIConnectionInfo_t | | |
|---|---|---|
| Wi-Fi info of the connected AP | | |
| Data Fields | | |
| uint8_t | ucSSID[wificonfigMAX_SSID_LEN] | SSID of the Wi-Fi network (binary array, not C-string) |
| uint8_t | ucSSIDLength | SSID length. |
| uint8_t | ucBSSID[wificonfigMAX_BSSID_LEN] | BSSID of the Wi-Fi network (binary array, not C-string) |
| WIFISecurity_t | xSecurity | Wi-Fi Security. |
| uint8_t | ucChannel | Channel info. |

### ◆ WiFiEventInfoReady_t

| struct WiFiEventInfoReady_t |
|---|
| Wi-Fi event info for WI-FI ready |

### ◆ WiFiEventInfoScanDone_t

| struct WiFiEventInfoScanDone_t |
|---|
| |

| Wi-Fi event info for scan done |
|---|

### ◆ WiFiEventInfoConnected_t

| struct WiFiEventInfoConnected_t |
|---|
| Wi-Fi event info for station connected to AP |

### ◆ WiFiEventInfoDisconnected_t

| struct WiFiEventInfoDisconnected_t | | |
|---|---|---|
| Wi-Fi event info for station disconnected from AP | | |
| Data Fields | | |
| WIFIReason_t | xReason | Reason code for station disconnection. |

### ◆ WiFiEventInfoConnectionFailed_t

| struct WiFiEventInfoConnectionFailed_t | | |
|---|---|---|
| Wi-Fi event info for station connection failure | | |
| Data Fields | | |
| WIFIReason_t | xReason | Reason code for connection failure. |

### ◆ WiFiEventInfoIPReady_t

| struct WiFiEventInfoIPReady_t | | |
|---|---|---|
| Wi-Fi event info for IP ready | | |
| Data Fields | | |
| WIFIIPAddress_t | xIPAddress | Station IP address from DHCP. |

### ◆ WiFiEventInfoAPStateChanged_t

| struct WiFiEventInfoAPStateChanged_t | | |
|---|---|---|
| Wi-Fi event info for AP state change | | |
| Data Fields | | |
| uint8_t | ucState | AP state: 0 = DOWN, 1 = UP. |

### ◆ WiFiEventInfoAPStationConnected_t

| struct WiFiEventInfoAPStationConnected_t | | |
|---|---|---|
| Wi-Fi event info for AP got a connected station | | |
| Data Fields | | |
| uint8_t | ucMac[wificonfigMAX_BSSID_LEN] | MAC address of connected station. |

### ◆ WiFiEventInfoAPStationDisconnected_t

| struct WiFiEventInfoAPStationDisconnected_t | | |
|---|---|---|
| Wi-Fi event info for AP got a disconnected station | | |
| Data Fields | | |
| uint8_t | ucMac[wificonfigMAX_BSSID_LEN] | MAC address of disconnected station. |
| WIFIReason_t | xReason | Reason code for the disconnection. |

### ◆ WiFiEventInfoRxDone_t

| struct WiFiEventInfoRxDone_t | | |
|---|---|---|
| Wi-Fi event info for receiving a frame in monitor mode (or normal mode with RX filter) | | |
| Data Fields | | |
| uint8_t * | pucData | Data buffer of received raw frame. |
| uint32_t | ulLength | Length of the raw frame. |

### ◆ WiFiEventInfoTxDone_t

| struct WiFiEventInfoTxDone_t |
|---|
| Wi-Fi event info for finishing transmitting an injection frame |

### ◆ WIFIEvent_t

| struct WIFIEvent_t |
|---|
| Wi-Fi combined event data structure |

### ◆ WIFIStatisticInfo_t

| struct WIFIStatisticInfo_t | | |
|---|---|---|
| Wi-Fi Statistic info | | |
| Data Fields | | |
| uint32_t | ulTxSuccessCount | Number of TX successes, 0 if unavailable. |
| uint32_t | ulTxRetryCount | Number of TX retries, 0 if unavailable. |
| uint32_t | ulTxFailCount | Number of TX failures, 0 if unavailable. |
| uint32_t | ulRxSuccessCount | Number of RX successes, 0 if unavailable. |
| uint32_t | ulRxCRCErrorCount | Number of RX FCS errors, 0 if unavailable. |

| | | |
|---|---|---|
| uint32_t | ulMICErrorCount | Number of MIC errors, 0 if unavailable. |
| int8_t | cNoise | Average noise level in dBm, -128 if unavailable. |
| uint16_t | usPhyRate | Maximum used PHY rate, 0 if unavailable. |
| uint16_t | usTxRate | Average used TX rate, 0 if unavailable. |
| uint16_t | usRxRate | Average used RX rate, 0 if unavailable. |
| int8_t | cRssi | Average beacon RSSI in dBm, -128 if unavailable. |
| uint8_t | ucBandwidth | Average used bandwidth, 0 if unavailable. |
| uint8_t | ucIdleTimePer | Percent of idle time, 0 if unavailable. |

### ◆ WIFICapabilityInfo_t

| struct WIFICapabilityInfo_t | | |
|---|---|---|
| Wi-Fi capabilities | | |
| Data Fields | | |
| WIFIBand_t | xBand | Supported band. |
| WIFIPhyMode_t | xPhyMode | Supported PHY mode. |
| WIFIBandwidth_t | xBandwidth | Supported bandwidth. |
| uint32_t | ulMaxAggr | Max aggregation length, 0 if no aggregation support. |
| uint16_t | usSupportedFeatures | Supported features bitmap, e.g., WIFI_WPS_SUPPORTED. |

## Macro Definition Documentation

### ◆ WIFI_WPS_SUPPORTED

#define WIFI_WPS_SUPPORTED

Wi-Fi lower level supported feature mask.

**See also**
WIFICapabilityInfo_t.

## Typedef Documentation

### ◆ WIFIEventHandler_t

typedef void(* WIFIEventHandler_t) (WIFIEvent_t *xEvent)

Wi-Fi event handler definition.

**Parameters**

| | | | |
|---|---|---|---|
| | [in] | xEvent | - Wi-Fi event data structure. |

**Returns**

None.

## Enumeration Type Documentation

### ◆ WIFIReturnCode_t

enum WIFIReturnCode_t

Return code denoting API status.

| Enumerator | |
|---|---|
| eWiFiSuccess | Success. |
| eWiFiFailure | Failure. |
| eWiFiTimeout | Timeout. |
| eWiFiNotSupported | Not supported. |

### ◆ WIFISecurity_t

| enum WIFISecurity_t | |
|---|---|
| Wi-Fi Security types | |
| Enumerator | |
| eWiFiSecurityOpen | Open - No Security. |
| eWiFiSecurityWEP | WEP Security. |
| eWiFiSecurityWPA | WPA Security. |
| eWiFiSecurityWPA2 | WPA2 Security. |
| eWiFiSecurityWPA2_ent | WPA2 Enterprise Security. |
| eWiFiSecurityWPA3 | WPA3 Security. |
| eWiFiSecurityNotSupported | Unknown Security. |

### ◆ WIFIDeviceMode_t

| enum WIFIDeviceMode_t | |
|---|---|
| Wi-Fi device modes | |
| Enumerator | |
| eWiFiModeStation | Station mode. |
| eWiFiModeAP | Access point mode. |
| eWiFiModeP2P | P2P mode. |
| eWiFiModeAPStation | AP+Station (repeater) mode. |
| eWiFiModeNotSupported | Unsupported mode. |

### ◆ WIFIPMMode_t

| enum WIFIPMMode_t | |
|---|---|
| Wi-Fi device power management modes | |
| Enumerator | |
| eWiFiPMNormal | Normal mode. |
| eWiFiPMLowPower | Low Power mode. |
| eWiFiPMAlwaysOn | Always On mode. |
| eWiFiPMNotSupported | Unsupported PM mode. |

### ◆ WIFIIPAddressType_t

| enum WIFIIPAddressType_t |
|---|
| Wi-Fi station IP address type |

### ◆ WIFIReason_t

| enum WIFIReason_t | |
|---|---|
| Wi-Fi protocol reason codes | |
| Enumerator | |
| eWiFiReasonUnspecified | Unspecified error. |
| eWiFiReasonAPNotFound | Cannot find the target AP. |
| eWiFiReasonAuthExpired | Previous auth invalid. |
| eWiFiReasonAuthLeaveBSS | Deauth leaving BSS. |
| eWiFiReasonAuthFailed | All other AUTH errors. |
| eWiFiReasonAssocExpired | Disassoc due to inactivity. |
| eWiFiReasonAssocTooMany | AP is overloaded. |
| eWiFiReasonAssocPowerCapBad | Power capability unacceptable. |
| eWiFiReasonAssocSupChanBad | Supported channel unacceptable. |
| eWiFiReasonAssocFailed | |

| | |
|---|---|
| | All other ASSOC errors. |
| eWiFiReasonIEInvalid | Management frame IE invalid. |
| eWiFiReasonMICFailure | MIC failure detected. |
| eWiFiReason4WayTimeout | 4WAY handshake timeout |
| eWiFiReason4WayIEDiffer | 4WAY handshake IE error |
| eWiFiReason4WayFailed | All other 4WAY errors. |
| eWiFiReasonAKMPInvalid | AKMP invalid. |
| eWiFiReasonPairwiseCipherInvalid | Pairwise cipher invalid. |
| eWiFiReasonGroupCipherInvalid | Group cipher invalid. |
| eWiFiReasonRSNVersionInvalid | RSN version invalid. |
| eWiFiReasonRSNCapInvalid | RSN capability invalid. |
| eWiFiReasonGroupKeyUpdateTimeout | Group key update timeout. |
| eWiFiReasonCipherSuiteRejected | Cipher violates security policy. |
| eWiFiReason8021XAuthFailed | 802.1X auth errors |
| eWiFiReasonBeaconTimeout | Beacon timeout. |
| eWiFiReasonLinkFailed | All other link errors. |
| eWiFiReasonDHCPExpired | DHCP license expired. |
| eWiFiReasonDHCPFailed | All other DHCP errors. |

### ◆ WIFIEventType_t

| enum WIFIEventType_t | |
|---|---|
| Wi-Fi event types | |
| Enumerator | |
| eWiFiEventReady | Wi-Fi is initialized or was reset in the lower layer. |
| eWiFiEventScanDone | Scan is finished. |
| eWiFiEventConnected | Station is connected to the AP. |
| eWiFiEventDisconnected | Station is disconnected from the AP. |
| eWiFiEventConnectionFailed | Station connection has failed. |
| eWiFiEventIPReady | DHCP is successful. |
| eWiFiEventAPStateChanged | SoftAP state changed. |
| eWiFiEventAPStationConnected | SoftAP got a new station. |
| eWiFiEventAPStationDisconnected | SoftAP lost a new station. |
| eWiFiEventWPSSuccess | WPS is completed successfully. |
| eWiFiEventWPSFailed | WPS has failed. |
| eWiFiEventWPSTimeout | WPS has timeout. |

### ◆ WIFIBand_t

| enum WIFIBand_t | |
|---|---|
| Wi-Fi band | |
| Enumerator | |
| eWiFiBand2G | 2.4G band |
| eWiFiBand5G | 5G band |
| eWiFiBandDual | Dual band. |
| eWiFiBandMax | Unsupported. |

### ◆ WIFIPhyMode_t

| enum WIFIPhyMode_t | |
|---|---|
| Wi-Fi PHY mode | |
| Enumerator | |
| eWiFiPhy11b | 11B |
| eWiFiPhy11g | 11G |
| eWiFiPhy11n | 11N |
| eWiFiPhy11ac | 11AC |
| eWiFiPhy11ax | 11AX |
| eWiFiPhyMax | Unsupported. |

### ◆ WIFIBandwidth_t

| enum WIFIBandwidth_t | |
|---|---|
| Wi-Fi bandwidth | |
| Enumerator | |
| eWiFiBW20 | 20MHz only |
| eWiFiBW40 | Max 40MHz. |
| eWiFiBW80 | Max 80MHz. |
| eWiFiBW160 | Max 80+80 or 160MHz. |
| eWiFiBWMax | Unsupported. |

## Function Documentation

### ◆ WIFI_On()

WIFIReturnCode_t WIFI_On ( void )

Turns on Wi-Fi.

This function turns on Wi-Fi module,initializes the drivers and must be called before calling any other Wi-Fi API

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned on, failure code otherwise.

Turns on Wi-Fi.

This function turns on Wi-Fi module,initializes the drivers and must be called before calling any other Wi-Fi API

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned on, failure code otherwise.

### ◆ WIFI_Off()

WIFIReturnCode_t WIFI_Off ( void )

Turns off Wi-Fi.

This function turns off the Wi-Fi module. The Wi-Fi peripheral should be put in a low power or off state in this routine.

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned off, failure code otherwise.

Turns off Wi-Fi.

This function turns off the Wi-Fi module. The Wi-Fi peripheral should be put in a low power or off state in this routine.

**Returns**

eWiFiSuccess if Wi-Fi module was successfully turned off, failure code otherwise.

### ◆ WIFI_ConnectAP()

WIFIReturnCode_t WIFI_ConnectAP ( const WIFINetworkParams_t *const *pxNetworkParams*)

Connects to the Wi-Fi Access Point (AP) specified in the input.

The Wi-Fi should stay connected when the same Access Point it is currently connected to is specified. Otherwise, the Wi-Fi should disconnect and connect to the new Access Point specified. If the new Access Point specified has invalid parameters, then the Wi-Fi should be disconnected.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkParams | Configuration to join AP. |

**Returns**

eWiFiSuccess if connection is successful, failure code otherwise.

**Example**

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;
xNetworkParams.pcSSID = "SSID String";
xNetworkParams.ucSSIDLength = SSIDLen;
xNetworkParams.pcPassword = "Password String";
xNetworkParams.ucPasswordLength = PassLength;
xNetworkParams.xSecurity = eWiFiSecurityWPA2;
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );
if(xWifiStatus == eWiFiSuccess)
{
 //Connected to AP.
}
```

**See also**

WIFINetworkParams_t

Connects to the Wi-Fi Access Point (AP) specified in the input.

The Wi-Fi should stay connected when the same Access Point it is currently connected to is specified. Otherwise, the Wi-Fi should disconnect and connect to the new Access Point specified. If the new Access Point specified has invalid parameters, then the Wi-Fi should be disconnected.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkParams | Configuration to join AP. |

**Returns**

eWiFiSuccess if connection is successful, failure code otherwise.

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;
```

```
xNetworkParams.pcSSID = "SSID String";
xNetworkParams.ucSSIDLength = SSIDLen;
xNetworkParams.pcPassword = "Password String";
xNetworkParams.ucPasswordLength = PassLength;
xNetworkParams.xSecurity = eWiFiSecurityWPA2;
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );
if(xWifiStatus == eWiFiSuccess)
{
 //Connected to AP.
}
```

**See also**
WIFINetworkParams_t

### ◆ WIFI_Disconnect()

WIFIReturnCode_t WIFI_Disconnect ( void )

Disconnects from the currently connected Access Point.

**Returns**
eWiFiSuccess if disconnection was successful or if the device is already disconnected, failure code otherwise.

Disconnects from the currently connected Access Point.

**Returns**
eWiFiSuccess if disconnection was successful or if the device is already disconnected, failure code otherwise.

### ◆ WIFI_Reset()

WIFIReturnCode_t WIFI_Reset ( void )

Resets the Wi-Fi Module.

**Returns**
eWiFiSuccess if Wi-Fi module was successfully reset, failure code otherwise.

Resets the Wi-Fi Module.

**Returns**
eWiFiSuccess if Wi-Fi module was successfully reset, failure code otherwise.

### ◆ WIFI_SetMode()

WIFIReturnCode_t WIFI_SetMode ( WIFIDeviceMode_t *xDeviceMode*)

Sets the Wi-Fi mode.

**Parameters**

| | | |
|---|---|---|
| [in] | xDeviceMode | - Mode of the device Station / Access Point /P2P. |

**Example**

```
WIFIReturnCode_t xWifiStatus;
xWifiStatus = WIFI_SetMode(eWiFiModeStation);
if(xWifiStatus == eWiFiSuccess)
{
 //device Set to station mode
}
```

**Returns**

eWiFiSuccess if Wi-Fi mode was set successfully, failure code otherwise.

### ◆ WIFI_GetMode()

WIFIReturnCode_t WIFI_GetMode ( WIFIDeviceMode_t * *pxDeviceMode*)

Gets the Wi-Fi mode.

**Parameters**

| | | |
|---|---|---|
| [out] | pxDeviceMode | - return mode Station / Access Point /P2P |

**Example**

```
WIFIReturnCode_t xWifiStatus;
WIFIDeviceMode_t xDeviceMode;
xWifiStatus = WIFI_GetMode(&xDeviceMode);
if(xWifiStatus == eWiFiSuccess)
{
 //device mode is xDeviceMode
}
```

**Returns**

eWiFiSuccess if Wi-Fi mode was successfully retrieved, failure code otherwise.

### ◆ WIFI_NetworkAdd()

WIFIReturnCode_t WIFI_NetworkAdd ( const WIFINetworkProfile_t *const *pxNetworkProfile*, uint16_t * *pusIndex* )

Add a Wi-Fi Network profile.

Adds a Wi-fi network to the network list in Non Volatile memory.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkProfile | - Network profile parameters |
| [out] | pusIndex | - Network profile index in storage |

**Returns**

Index of the profile storage on success, or failure return code on failure.

**Example**

```
WIFINetworkProfile_t xNetworkProfile = {0};
WIFIReturnCode_t xWiFiStatus;
uint16_t usIndex;
strncpy( xNetworkProfile.cSSID, "SSID_Name", SSIDLen));
xNetworkProfile.ucSSIDLength = SSIDLen;
strncpy( xNetworkProfile.cPassword, "PASSWORD",PASSLen );
xNetworkProfile.ucPasswordLength = PASSLen;
xNetworkProfile.xSecurity = eWiFiSecurityWPA2;
WIFI_NetworkAdd( &xNetworkProfile, &usIndex );
```

### ◆ WIFI_NetworkGet()

WIFIReturnCode_t WIFI_NetworkGet ( WIFINetworkProfile_t * *pxNetworkProfile*, uint16_t *usIndex* )

Get a Wi-Fi network profile.

Gets the Wi-Fi network parameters at the given index from network list in non-volatile memory.

***Note***

*The WIFINetworkProfile_t data returned must have the the SSID and Password lengths specified as the length without a null terminator.*

**Parameters**

| | | |
|---|---|---|
| [out] | pxNetworkProfile | - pointer to return network profile parameters |
| [in] | usIndex | - Index of the network profile, must be between 0 to wificonfigMAX_NETWORK_PROFILES |

**Returns**

eWiFiSuccess if the network profile was successfully retrieved, failure code otherwise.

**See also**

WIFINetworkProfile_t

**Example**

```
WIFINetworkProfile_t xNetworkProfile = {0};

uint16_t usIndex = 3; //Get profile stored at index 3.

WIFI_NetworkGet( &xNetworkProfile, usIndex );
```

### ◆ WIFI_NetworkDelete()

WIFIReturnCode_t WIFI_NetworkDelete ( uint16_t *usIndex*)

Delete a Wi-Fi Network profile.

Deletes the Wi-Fi network profile from the network profile list at given index in non-volatile memory

**Parameters**

| | | |
|---|---|---|
| [in] | usIndex | - Index of the network profile, must be between 0 to wificonfigMAX_NETWORK_PROFILES. |

If wificonfigMAX_NETWORK_PROFILES is the index, then all network profiles will be deleted.

**Returns**

eWiFiSuccess if successful, failure code otherwise. If successful, the interface IP address is copied into the IP address buffer.

**Example**

```
uint16_t usIndex = 2; //Delete profile at index 2

WIFI_NetworkDelete( usIndex );
```

### ◆ WIFI_Ping()

WIFIReturnCode_t WIFI_Ping ( uint8_t * *pucIPAddr*, uint16_t *usCount*, uint32_t *ulIntervalMS* )

Ping an IP address in the network.

**Parameters**

| | | |
|---|---|---|
| [in] | pucIPAddr | IP Address array to ping. |
| [in] | usCount | Number of times to ping |
| [in] | ulIntervalMS | Interval in mili-seconds for ping operation |

**Returns**

eWiFiSuccess if ping was successful, other failure code otherwise.

Ping an IP address in the network.

**Parameters**

| | | |
|---|---|---|
| [in] | pucIPAddr | IP Address array to ping. |
| [in] | usCount | Number of times to ping |
| [in] | ulIntervalMS | Interval in milliseconds for ping operation |

**Returns**

eWiFiSuccess if ping was successful, other failure code otherwise.

### ◆ WIFI_GetIPInfo()

WIFIReturnCode_t WIFI_GetIPInfo ( WIFIIPConfiguration_t * *pxIPInfo*)

Get IP configuration (IP address, NetworkMask, Gateway and DNS server addresses).

**Parameters**

| | | |
|---|---|---|
| [out] | pxIPInfo | - Current IP configuration. |

**Returns**

eWiFiSuccess if successful and IP Address buffer has the interface's IP address, failure code otherwise.

**Example**

```
WIFIIPConfiguration_t xIPInfo;

WIFI_GetIPInfo( &xIPInfo );
```

This is a synchronous call.

**Parameters**

| | | |
|---|---|---|
| [out] | pxIPInfo | - Current IP configuration. |

**Returns**

eWiFiSuccess if connection info was got successfully, failure code otherwise.

**Parameters**

| | | |
|---|---|---|
| [out] | pxIPInfo | - Current IP configuration. |

**Returns**

eWiFiSuccess if successful and IP Address buffer has the interface's IP address, failure code otherwise.

**Example**

```
WIFIIPConfiguration_t xIPInfo;

WIFI_GetIPInfo( &xIPInfo );
```

This is a synchronous call.

**Parameters**

| | | |
|---|---|---|
| [out] | pxIPInfo | - Current IP configuration. |

**Returns**

eWiFiSuccess if connection info was got successfully, failure code otherwise.

**Parameters**

| | | |
|---|---|---|
| [out] | pxIPInfo | - Current IP configuration. |

**Returns**

eWiFiSuccess if successful and IP Address buffer has the interface's IP address, failure code otherwise.

**Example**

```
WIFIIPConfiguration_t xIPInfo;

WIFI_GetIPInfo( &xIPInfo );
```

### ◆ WIFI_GetMAC()

WIFIReturnCode_t WIFI_GetMAC ( uint8_t * *pucMac*)

Retrieves the Wi-Fi interface's MAC address.

**Parameters**

| | | |
|---|---|---|
| [out] | pucMac | MAC Address buffer sized 6 bytes. |

**Example**

```
uint8_t ucMacAddressVal[ wificonfigMAX_BSSID_LEN ];

WIFI_GetMAC( &ucMacAddressVal[0] );
```

**Returns**

eWiFiSuccess if the MAC address was successfully retrieved, failure code otherwise. The returned MAC address must be 6 consecutive bytes with no delimitters.

Retrieves the Wi-Fi interface's MAC address.

**Parameters**

| | | |
|---|---|---|
| [out] | pucMac | MAC Address buffer sized 6 bytes. |

```
uint8_t ucMacAddressVal[ wificonfigMAX_BSSID_LEN ];

WIFI_GetMAC( &ucMacAddressVal[0] );
```

**Returns**

eWiFiSuccess if the MAC address was successfully retrieved, failure code otherwise. The returned MAC address must be 6 consecutive bytes with no delimitters.

### ◆ WIFI_GetHostIP()

WIFIReturnCode_t WIFI_GetHostIP ( char * *pcHost*, uint8_t * *pucIPAddr* )

Retrieves the host IP address from a host name using DNS.

**Parameters**

| | | |
|---|---|---|
| [in] | pcHost | - Host (node) name. |
| [in] | pucIPAddr | - IP Address buffer. |

**Returns**

eWiFiSuccess if the host IP address was successfully retrieved, failure code otherwise.

**Example**

```
uint8_t ucIPAddr[ 4 ];

WIFI_GetHostIP( "amazon.com", &ucIPAddr[0] );
```

Retrieves the host IP address from a host name using DNS.

**Parameters**

| | | |
|---|---|---|
| [in] | pcHost | - Host (node) name. |
| [in] | pucIPAddr | - IP Address buffer. |

**Returns**

eWiFiSuccess if the host IP address was successfully retrieved, failure code otherwise.

```
uint8_t ucIPAddr[ 4 ];

WIFI_GetHostIP( "amazon.com", &ucIPAddr[0] );
```

### ◆ WIFI_Scan()

WIFIReturnCode_t WIFI_Scan ( WIFIScanResult_t * *pxBuffer*, uint8_t *ucNumNetworks* )

Perform a Wi-Fi network Scan.

**Parameters**

| | | |
|---|---|---|
| [in] | pxBuffer | - Buffer for scan results. |
| [in] | ucNumNetworks | - Number of networks to retrieve in scan result. |

**Returns**

eWiFiSuccess if the Wi-Fi network scan was successful, failure code otherwise.

***Note***

*The input buffer will have the results of the scan.*

**Example**

```
const uint8_t ucNumNetworks = 10; //Get 10 scan results
WIFIScanResult_t xScanResults[ ucNumNetworks ];
WIFI_Scan( xScanResults, ucNumNetworks );
```

Perform a Wi-Fi network Scan.

**Parameters**

| | | |
|---|---|---|
| [in] | pxBuffer | - Buffer for scan results. |
| [in] | ucNumNetworks | - Number of networks to retrieve in scan result. |

**Returns**

eWiFiSuccess if the Wi-Fi network scan was successful, failure code otherwise.

***Note***

*The input buffer will have the results of the scan.*

```
const uint8_t ucNumNetworks = 10; //Get 10 scan results
WIFIScanResult_t xScanResults[ ucNumNetworks ];
WIFI_Scan( xScanResults, ucNumNetworks );
```

### ◆ WIFI_StartAP()

WIFIReturnCode_t WIFI_StartAP ( void )

Start SoftAP mode.

**Returns**

eWiFiSuccess if SoftAP was successfully started, failure code otherwise.

### ◆ WIFI_StopAP()

WIFIReturnCode_t WIFI_StopAP ( void )

Stop SoftAP mode.

**Returns**

eWiFiSuccess if the SoftAP was successfully stopped, failure code otherwise.

### ◆ WIFI_ConfigureAP()

WIFIReturnCode_t WIFI_ConfigureAP ( const WIFINetworkParams_t *const *pxNetworkParams*)

Configure SoftAP.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkParams | - Network parameters to configure AP. |

**Returns**

eWiFiSuccess if SoftAP was successfully configured, failure code otherwise.

**Example**

```
WIFINetworkParams_t xNetworkParams;
xNetworkParams.pcSSID = "SSID_Name";
xNetworkParams.pcPassword = "PASSWORD";
xNetworkParams.xSecurity = eWiFiSecurityWPA2;
xNetworkParams.cChannel = ChannelNum;
WIFI_ConfigureAP( &xNetworkParams );
```

### ◆ WIFI_SetPMMode()

WIFIReturnCode_t WIFI_SetPMMode ( WIFIPMMode_t *xPMModeType*, const void * *pvOptionValue* )

Set the Wi-Fi power management mode.

**Parameters**

| | | |
|---|---|---|
| [in] | xPMModeType | - Power mode type. |
| [in] | pvOptionValue | - A buffer containing the value of the option to set depends on the mode type example - beacon interval in sec |

**Returns**

eWiFiSuccess if the power mode was successfully configured, failure code otherwise.

### ◆ WIFI_GetPMMode()

WIFIReturnCode_t WIFI_GetPMMode ( WIFIPMMode_t * *pxPMModeType*, void * *pvOptionValue* )

Get the Wi-Fi power management mode.

**Parameters**

| | | |
|---|---|---|
| [out] | pxPMModeType | - pointer to get current power mode set. |
| [out] | pvOptionValue | - optional value |

**Returns**

eWiFiSuccess if the power mode was successfully retrieved, failure code otherwise.

### ◆ WIFI_RegisterEvent()

WIFIReturnCode_t WIFI_RegisterEvent ( WIFIEventType_t *xEventType*, WIFIEventHandler_t *xHandler* )

Register a Wi-Fi event Handler.

**Parameters**

| | | |
|---|---|---|
| [in] | xEventType | - Wi-Fi event type. |
| [in] | xHandler | - Wi-Fi event handler. |

**Returns**

eWiFiSuccess if registration is successful, failure code otherwise.

### ◆ WIFI_IsConnected()

WIFIReturnCode_t WIFI_IsConnected ( const WIFINetworkParams_t * *pxNetworkParams*)

Check if the Wi-Fi is connected and the AP configuration matches the query.

param[in] pxNetworkParams - Network parameters to query, if NULL then just check the Wi-Fi link status.

### ◆ WIFI_StartScan()

WIFIReturnCode_t WIFI_StartScan ( WIFIScanConfig_t * *pxScanConfig*)

Start a Wi-Fi scan.

This is an asynchronous call, the result will be notified by an event.

**See also**

WiFiEventInfoScanDone_t.

**Parameters**

| | | |
|---|---|---|
| [in] | pxScanConfig | - Parameters for scan, NULL if default scan (i.e. broadcast scan on all channels). |

**Returns**

eWiFiSuccess if scan was started successfully, failure code otherwise.

### ◆ WIFI_GetScanResults()

WIFIReturnCode_t WIFI_GetScanResults ( const WIFIScanResult_t ** *pxBuffer*, uint16_t * *ucNumNetworks* )

Get Wi-Fi scan results. It should be called only after scan is completed. Scan results are sorted in decreasing rssi order.

**Parameters**

| | | |
|---|---|---|
| [out] | pxBuffer | - Handle to the READ ONLY scan results buffer. |
| [out] | ucNumNetworks | - Actual number of networks in the scan results. |

**Returns**

eWiFiSuccess if the scan results were got successfully, failure code otherwise.

### ◆ WIFI_StartConnectAP()

WIFIReturnCode_t WIFI_StartConnectAP ( const WIFINetworkParams_t * *pxNetworkParams*)

Connect to the Wi-Fi Access Point (AP) specified in the input.

This is an asynchronous call, the result will be notified by an event.

**See also**
WiFiEventInfoConnected_t.

The Wi-Fi should stay connected when the specified AP is the same as the currently connected AP. Otherwise, the Wi-Fi should disconnect and connect to the specified AP. If the specified AP has invalid parameters, the Wi-Fi should be disconnected.

**Parameters**

| | | |
|---|---|---|
| [in] | pxNetworkParams | - Configuration of the targeted AP. |

**Returns**
eWiFiSuccess if connection was started successfully, failure code otherwise.

### ◆ WIFI_StartDisconnect()

WIFIReturnCode_t WIFI_StartDisconnect ( void )

Wi-Fi station disconnects from AP.

This is an asynchronous call. The result will be notified by an event.

**See also**
WiFiEventInfoDisconnected_t.

**Returns**
eWiFiSuccess if disconnection was started successfully, failure code otherwise.

### ◆ WIFI_GetConnectionInfo()

WIFIReturnCode_t WIFI_GetConnectionInfo ( WIFIConnectionInfo_t * *pxConnectionInfo*)

Get Wi-Fi info of the connected AP.

This is a synchronous call.

**Parameters**

| | | |
|---|---|---|
| [out] | pxConnectionInfo | - Wi-Fi info of the connected AP. |

**Returns**
eWiFiSuccess if connection info was got successfully, failure code otherwise.

### ◆ WIFI_GetRSSI()

WIFIReturnCode_t WIFI_GetRSSI ( int8_t * *pcRSSI*)

Get the RSSI of the connected AP.

This only works when the station is connected.

**Parameters**

| | | |
|---|---|---|
| [out] | pcRSSI | - RSSI of the connected AP. |

**Returns**

eWiFiSuccess if RSSI was got successfully, failure code otherwise.

### ◆ WIFI_GetStationList()

WIFIReturnCode_t WIFI_GetStationList ( WIFIStationInfo_t * *pxStationList*, uint8_t * *pcStationListSize* )

SoftAP mode get connected station list.

**Parameters**

| | | |
|---|---|---|
| [out] | pxStationList | - Buffer for station list, supplied by the caller. |
| [in,out] | pcStationListSize | - Input size of the list, output number of connected stations. |

**Returns**

eWiFiSuccess if stations were got successfully (manybe none), failure code otherwise.

### ◆ WIFI_StartDisconnectStation()

WIFIReturnCode_t WIFI_StartDisconnectStation ( uint8_t * *pucMac*)

AP mode disconnecting a station.

This is an asynchronous call, the result will be notified by an event.

**See also**

WiFiEventInfoAPStationDisconnected_t.

**Parameters**

| | | |
|---|---|---|
| [in] | pucMac | - MAC Address of the station to be disconnected. |

**Returns**

eWiFiSuccess if disconnection was started successfully, failure code otherwise.

### ◆ WIFI_SetSpoofingMAC()

WIFIReturnCode_t WIFI_SetSpoofingMAC ( uint8_t * *pucMac*)

Set Wi-Fi MAC addresses.

The given MAC address is for MAC SPOOFING. The original MAC address is not changed. This MAC address can be set only for Station. The AP MAC address (i.e. BSSID) will not be changed. If the original MAC address needs to be used, then MAC spoofing needs to be disabled.

**Parameters**

| | | |
|---|---|---|
| [in] | pucMac | - Station MAC address. |

**Returns**

eWiFiSuccess if MAC address was set successfully, failure code otherwise.

***Note***

*This function performs MAC SPOOFING, it does not change the "manufacturer" WIFI MAC address.*
*On some platforms the change of MAC address can only take effect after reboot.*

### ◆ WIFI_SetCountryCode()

WIFIReturnCode_t WIFI_SetCountryCode ( const char * *pcCountryCode*)

Set country based configuration (including channel list, power table)

**Parameters**

| | | |
|---|---|---|
| [in] | pcCountryCode | - Country code (null-terminated string, e.g. "US", "CN". See ISO-3166). |

**Returns**

eWiFiSuccess if Country Code is set successfully, failure code otherwise.

### ◆ WIFI_GetCountryCode()

WIFIReturnCode_t WIFI_GetCountryCode ( char * *pcCountryCode*)

Get the currently configured country code.

**Parameters**

| | | |
|---|---|---|
| [out] | pcCountryCode | - Null-terminated string to hold the country code (see ISO-3166). Must be at least 4 bytes. |

**Returns**

eWiFiSuccess if Country Code is retrieved successfully, failure code otherwise.

### ◆ WIFI_GetStatistic()

WIFIReturnCode_t WIFI_GetStatistic ( WIFIStatisticInfo_t * *pxStats*)

Get the Wi-Fi statistics.

**Parameters**

| | | |
|---|---|---|
| [out] | pxStats | - Structure to hold the WiFi statistics. |

**Returns**

eWiFiSuccess if statistics are retrieved successfully, failure code otherwise.

### ◆ WIFI_GetCapability()

WIFIReturnCode_t WIFI_GetCapability ( WIFICapabilityInfo_t * *pxCaps*)

Get the Wi-Fi capability.

**Parameters**

| | | |
|---|---|---|
| [out] | pxCaps | - Structure to hold the Wi-Fi capabilities. |

**Returns**

eWiFiSuccess if capabilities are retrieved successfully, failure code otherwise.

## 5.3.12 Power

Interfaces

### Detailed Description

Power Interfaces.

### Modules

Low Power Modes Interface

Interface for accessing low power modes.

### 5.3.12.1 Low Power Modes Interface

Interfaces » Power

### Detailed Description

Interface for accessing low power modes.

# Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPM driver supports configuration of MCU low power modes using the LPM hardware peripheral. The LPM driver supports low power modes deep standby, standby, sleep, and snooze.

*Note*

*Not all low power modes are available on all MCUs.*

### Data Structures

| | |
|---|---|
| struct | lpm_ram_retention_t |
| struct | lpm_ldo_standby_cfg_t |
| struct | lpm_cfg_t |
| struct | lpm_api_t |
| struct | lpm_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | lpm_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | lpm_mode_t |
| enum | lpm_snooze_request_t |
| enum | lpm_snooze_end_t |
| enum | lpm_snooze_cancel_t |
| enum | lpm_snooze_dtc_t |
| enum | lpm_standby_wake_source_t |
| enum | lpm_io_port_t |
| enum | lpm_power_supply_t |
| enum | lpm_deep_standby_cancel_edge_t |
| enum | lpm_deep_standby_soft_start_mode_t |

| | |
|---|---|
| enum | lpm_deep_standby_cancel_source_t |
| enum | lpm_output_port_enable_t |
| enum | lpm_ldo_standby_operation_t |
| enum | lpm_flash_mode_select_t |
| enum | lpm_hoco_startup_speed_t |
| enum | lpm_standby_sosc_t |

## Data Structure Documentation

### ◆ lpm_ram_retention_t

| struct lpm_ram_retention_t | | |
|---|---|---|
| RAM Retention Configuration for deep sleep and standby modes. | | |
| Data Fields | | |
| uint16_t | ram_retention | Configure RAM retention in software standby mode. |
| bool | tcm_retention | Enable or disable TCM retention in deep sleep and software standby modes. |

### ◆ lpm_ldo_standby_cfg_t

| struct lpm_ldo_standby_cfg_t | | |
|---|---|---|
| Configure LDO operation in standby mode. | | |
| Data Fields | | |
| lpm_ldo_standby_operation_t | pll1_ldo | Configure the state of PLL1 LDO in standby mode. |
| lpm_ldo_standby_operation_t | pll2_ldo | Configure the state of PLL2 LDO in standby mode. |
| lpm_ldo_standby_operation_t | hoco_ldo | Configure the state of HOCO LDO in standby mode. |

### ◆ lpm_cfg_t

| struct lpm_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| lpm_mode_t | low_power_mode | Low Power Mode |
| lpm_standby_wake_source_bits_ | standby_wake_sources | Bitwise list of sources to wake |

| | | |
|---|---|---|
| t | | from deep sleep and standby mode |
| lpm_snooze_request_t | snooze_request_source | Snooze request source |
| lpm_snooze_end_bits_t | snooze_end_sources | Bitwise list of snooze end sources |
| lpm_snooze_cancel_t | snooze_cancel_sources | List of snooze cancel sources |
| lpm_snooze_dtc_t | dtc_state_in_snooze | State of DTC in snooze mode, enabled or disabled |
| lpm_output_port_enable_t | output_port_enable | Output port enabled/disabled in standby and deep standby |
| lpm_io_port_t | io_port_state | IO port state in deep standby (maintained or reset) |
| lpm_power_supply_t | power_supply_state | Internal power supply state in standby and deep standby (deepcut) |
| lpm_deep_standby_cancel_source_bits_t | deep_standby_cancel_source | Sources that can trigger exit from deep standby |
| lpm_deep_standby_cancel_edge_bits_t | deep_standby_cancel_edge | Signal edges for the sources that can trigger exit from deep standby |
| lpm_ram_retention_t | ram_retention_cfg | RAM retention configuration for deep sleep and standby modes. |
| lpm_ldo_standby_cfg_t | ldo_standby_cfg | Configure LDOs that are disabled in standby mode. |
| void const * | p_extend | Placeholder for extension. |

### ◆ lpm_api_t

| struct lpm_api_t | |
|---|---|
| LPM driver structure. General LPM functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(lpm_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | lowPowerReconfigure )(lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | lowPowerModeEnter )(lpm_ctrl_t *const p_ctrl) |
| | |

| fsp_err_t(* | ioKeepClear )(lpm_ctrl_t *const p_ctrl) |
|---|---|

## Field Documentation

### ◆ open

fsp_err_t(* lpm_api_t::open) (lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)

Initialization function

### ◆ close

fsp_err_t(* lpm_api_t::close) (lpm_ctrl_t *const p_ctrl)

Initialization function

### ◆ lowPowerReconfigure

fsp_err_t(* lpm_api_t::lowPowerReconfigure) (lpm_ctrl_t *const p_ctrl, lpm_cfg_t const *const p_cfg)

Configure a low power mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ lowPowerModeEnter

fsp_err_t(* lpm_api_t::lowPowerModeEnter) (lpm_ctrl_t *const p_ctrl)

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

### ◆ ioKeepClear

fsp_err_t(* lpm_api_t::ioKeepClear) (lpm_ctrl_t *const p_ctrl)

Clear the IOKEEP bit after deep software standby.

### ◆ lpm_instance_t

struct lpm_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| lpm_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| lpm_cfg_t const *const | p_cfg | Pointer to the configuration structure for this instance. |
| lpm_api_t const *const | p_api | Pointer to the API structure for |

| | | |
|---|---|---|
| | | this instance. |

## Typedef Documentation

### ◆ lpm_ctrl_t

| typedef void lpm_ctrl_t |
|---|
| LPM control block. Allocate an instance specific control block to pass into the LPM API calls. |

## Enumeration Type Documentation

### ◆ lpm_mode_t

| enum lpm_mode_t | |
|---|---|
| Low power modes | |
| Enumerator | |
| LPM_MODE_SLEEP | Sleep mode. |
| LPM_MODE_DEEP_SLEEP | Deep Sleep mode. |
| LPM_MODE_STANDBY | Software Standby mode. |
| LPM_MODE_STANDBY_SNOOZE | Software Standby mode with Snooze mode enabled. |
| LPM_MODE_DEEP | Deep Software Standby mode. |

### ◆ lpm_snooze_request_t

| enum lpm_snooze_request_t | |
|---|---|
| Snooze request sources | |
| Enumerator | |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | |

| | |
|---|---|
| | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| | |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPHS0 | Enable High-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ8 | Enable IRQ8 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ9 | Enable IRQ9 pin snooze request. |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ10 | Enable IRQ10 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ11 | Enable IRQ11 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ12 | Enable IRQ12 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ13 | Enable IRQ13 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ14 | Enable IRQ14 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ15 | Enable IRQ15 pin snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ8 | Enable IRQ8 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ9 | Enable IRQ9 pin snooze request. |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ10 | Enable IRQ10 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ11 | Enable IRQ11 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ12 | Enable IRQ12 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ14 | Enable IRQ14 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ15 | Enable IRQ15 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ9 | Enable IRQ9 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ11 | Enable IRQ11 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ14 | Enable IRQ14 pin snooze request. |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ15 | Enable IRQ15 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPLP0 | Enable Low-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_RXD0_FALLING | Enable RXD0 falling edge snooze request. |
| LPM_SNOOZE_REQUEST_IRQ0 | Enable IRQ0 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ1 | Enable IRQ1 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ2 | Enable IRQ2 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ3 | Enable IRQ3 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ4 | Enable IRQ4 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ5 | Enable IRQ5 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ6 | Enable IRQ6 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ7 | Enable IRQ7 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ8 | Enable IRQ8 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ9 | Enable IRQ9 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ10 | Enable IRQ10 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ11 | Enable IRQ11 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ12 | Enable IRQ12 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ13 | Enable IRQ13 pin snooze request. |

| | |
|---|---|
| LPM_SNOOZE_REQUEST_IRQ14 | Enable IRQ14 pin snooze request. |
| LPM_SNOOZE_REQUEST_IRQ15 | Enable IRQ15 pin snooze request. |
| LPM_SNOOZE_REQUEST_KEY | Enable KR snooze request. |
| LPM_SNOOZE_REQUEST_ACMPHS0 | Enable High-speed analog comparator 0 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM1 | Enable RTC alarm 1 snooze request. |
| LPM_SNOOZE_REQUEST_RTC_ALARM | Enable RTC alarm snooze request. |
| LPM_SNOOZE_REQUEST_RTC_PERIOD | Enable RTC period snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW | Enable AGT1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGTW1_UNDERFLOW | Enable AGTW1 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_A | Enable AGT1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGTW1_COMPARE_A | Enable AGTW1 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT1_COMPARE_B | Enable AGT1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_AGTW1_COMPARE_B | Enable AGTW1 compare match B snooze request. |
| LPM_SNOOZE_REQUEST_AGT3_UNDERFLOW | Enable AGT3 underflow snooze request. |
| LPM_SNOOZE_REQUEST_AGT3_COMPARE_A | Enable AGT3 compare match A snooze request. |
| LPM_SNOOZE_REQUEST_AGT3_COMPARE_B | Enable AGT3 compare match B snooze request. |

### ◆ lpm_snooze_end_t

| enum lpm_snooze_end_t | |
|---|---|
| Snooze end control | |
| Enumerator | |
| LPM_SNOOZE_END_STANDBY_WAKE_SOURCES | Transition from Snooze to Normal mode directly. |
| LPM_SNOOZE_END_AGT1_UNDERFLOW | AGT1 underflow. |
| LPM_SNOOZE_END_DTC_TRANS_COMPLETE | Last DTC transmission completion. |
| LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED | Not Last DTC transmission completion. |
| LPM_SNOOZE_END_ADC0_COMPARE_MATCH | ADC Channel 0 compare match. |
| LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH | ADC Channel 0 compare mismatch. |
| LPM_SNOOZE_END_SCI0_ADDRESS_MISMATCH | SCI0 address mismatch. |
| LPM_SNOOZE_END_CTSU_NAJED | CTSU2SLa non-touch automatic judgment. |
| LPM_SNOOZE_END_STANDBY_WAKE_SOURCES | Transition from Snooze to Normal mode directly. |
| LPM_SNOOZE_END_AGT1_UNDERFLOW | AGT1 underflow. |
| LPM_SNOOZE_END_AGTW1_UNDERFLOW | AGTW1 underflow. |
| LPM_SNOOZE_END_DTC_TRANS_COMPLETE | Last DTC transmission completion. |
| LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED | Not Last DTC transmission completion. |
| LPM_SNOOZE_END_ADC0_COMPARE_MATCH | ADC Channel 0 compare match. |
| LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH | ADC Channel 0 compare mismatch. |
| LPM_SNOOZE_END_ADC1_COMPARE_MATCH | ADC 1 compare match. |
| LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH | ADC 1 compare mismatch. |
| LPM_SNOOZE_END_SCI0_ADDRESS_MISMATCH | SCI0 address mismatch. |
| LPM_SNOOZE_END_AGT3_UNDERFLOW | AGT3 underflow. |

### ◆ lpm_snooze_cancel_t

| enum lpm_snooze_cancel_t | |
|---|---|
| Snooze cancel control | |
| Enumerator | |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_SCI0_AM | SCI0 address match event. |
| LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE | DTC transfer completion. |
| LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI | Data operation circuit interrupt. |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_NONE | No snooze cancel source. |
| LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM | ADC Channel 0 window compare match. |
| LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM | ADC Channel 0 window compare mismatch. |
| LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE | DTC transfer completion. |
| LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI | Data operation circuit interrupt. |

### ◆ lpm_snooze_dtc_t

| enum lpm_snooze_dtc_t | |
|---|---|
| DTC Enable in Snooze Mode | |
| Enumerator | |
| LPM_SNOOZE_DTC_DISABLE | Disable DTC operation. |
| LPM_SNOOZE_DTC_ENABLE | Enable DTC operation. |

### ◆ lpm_standby_wake_source_t

| enum lpm_standby_wake_source_t | |
|---|---|
| Wake from deep sleep or standby mode sources, does not apply to sleep or deep standby modes | |
| Enumerator | |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IICA0 | I2CA 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTC | RTC interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ITL | Interval signal of 32-bit interval timer Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0ERRI | UARTA0 reception communication error Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0TXI | UARTA0 transmission transfer end or buffer empty Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0RXI | UARTA0 reception transfer end Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| | |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IICA0 | IICA0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTC | RTC interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ITL | Interval signal of 32-bit interval timer interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0ERRI | UARTA0 reception communication error interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0TXI | UARTA0 transmission transfer end or buffer empty interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0RXI | UARTA0 reception transfer end interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IICA1 | IICA1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1ERRI | UARTA1 reception communication error interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1TXI | UARTA1 transmission transfer end or buffer empty interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1RXI | UARTA1 reception transfer end interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IICA0 | IICA0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTC | RTC interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ITL | Interval signal of 32-bit interval timer interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0ERRI | UARTA0 reception communication error interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0TXI | UARTA0 transmission transfer end or buffer empty interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA0RXI | UARTA0 reception transfer end interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IICA1 | IICA1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1ERRI | UARTA1 reception communication error interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1TXI | UARTA1 transmission transfer end or buffer empty interrupt. |
| LPM_STANDBY_WAKE_SOURCE_UARTA1RXI | UARTA1 reception transfer end interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ACMPLP0 | Analog Comparator Low-speed 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBHS | USB High-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_COMPHS0 | Comparator-HS0 Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0U | ULPT0 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0A | ULPT0 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0B | ULPT0 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1U | ULPT1 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1A | ULPT1 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1B | ULPT1 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_PDM | PDM Sound Detection. |
| LPM_STANDBY_WAKE_SOURCE_IRQ16 | IRQ16. |
| LPM_STANDBY_WAKE_SOURCE_IRQ17 | IRQ17. |
| LPM_STANDBY_WAKE_SOURCE_IRQ18 | IRQ18. |
| LPM_STANDBY_WAKE_SOURCE_IRQ19 | IRQ19. |
| LPM_STANDBY_WAKE_SOURCE_IRQ20 | IRQ20. |
| LPM_STANDBY_WAKE_SOURCE_IRQ21 | IRQ21. |
| LPM_STANDBY_WAKE_SOURCE_IRQ22 | IRQ22. |
| LPM_STANDBY_WAKE_SOURCE_IRQ23 | IRQ23. |
| LPM_STANDBY_WAKE_SOURCE_IRQ24 | IRQ24. |
| LPM_STANDBY_WAKE_SOURCE_IRQ25 | IRQ25. |
| LPM_STANDBY_WAKE_SOURCE_IRQ26 | IRQ26. |
| LPM_STANDBY_WAKE_SOURCE_IRQ27 | IRQ27. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ28 | IRQ28. |
| LPM_STANDBY_WAKE_SOURCE_IRQ29 | IRQ29. |
| LPM_STANDBY_WAKE_SOURCE_IRQ30 | IRQ30. |
| LPM_STANDBY_WAKE_SOURCE_IRQ31 | IRQ31. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBHS | USB High-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_COMPHS0 | Comparator-HS0 Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0U | ULPT0 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0A | ULPT0 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0B | ULPT0 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1U | ULPT1 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1A | ULPT1 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1B | ULPT1 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_PDM | PDM Sound Detection. |
| LPM_STANDBY_WAKE_SOURCE_IRQ16 | IRQ16. |
| LPM_STANDBY_WAKE_SOURCE_IRQ17 | IRQ17. |
| LPM_STANDBY_WAKE_SOURCE_IRQ18 | IRQ18. |
| LPM_STANDBY_WAKE_SOURCE_IRQ19 | IRQ19. |
| LPM_STANDBY_WAKE_SOURCE_IRQ20 | IRQ20. |
| LPM_STANDBY_WAKE_SOURCE_IRQ21 | IRQ21. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ22 | IRQ22. |
| LPM_STANDBY_WAKE_SOURCE_IRQ23 | IRQ23. |
| LPM_STANDBY_WAKE_SOURCE_IRQ24 | IRQ24. |
| LPM_STANDBY_WAKE_SOURCE_IRQ25 | IRQ25. |
| LPM_STANDBY_WAKE_SOURCE_IRQ26 | IRQ26. |
| LPM_STANDBY_WAKE_SOURCE_IRQ27 | IRQ27. |
| LPM_STANDBY_WAKE_SOURCE_IRQ28 | IRQ28. |
| LPM_STANDBY_WAKE_SOURCE_IRQ29 | IRQ29. |
| LPM_STANDBY_WAKE_SOURCE_IRQ30 | IRQ30. |
| LPM_STANDBY_WAKE_SOURCE_IRQ31 | IRQ31. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBHS | USB High-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_COMPHS0 | Comparator-HS0 Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0U | ULPT0 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0A | ULPT0 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0B | ULPT0 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1U | ULPT1 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1A | ULPT1 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1B | ULPT1 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_PDM | PDM Sound Detection. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ16 | IRQ16. |
| LPM_STANDBY_WAKE_SOURCE_IRQ17 | IRQ17. |
| LPM_STANDBY_WAKE_SOURCE_IRQ18 | IRQ18. |
| LPM_STANDBY_WAKE_SOURCE_IRQ19 | IRQ19. |
| LPM_STANDBY_WAKE_SOURCE_IRQ20 | IRQ20. |
| LPM_STANDBY_WAKE_SOURCE_IRQ21 | IRQ21. |
| LPM_STANDBY_WAKE_SOURCE_IRQ22 | IRQ22. |
| LPM_STANDBY_WAKE_SOURCE_IRQ23 | IRQ23. |
| LPM_STANDBY_WAKE_SOURCE_IRQ24 | IRQ24. |
| LPM_STANDBY_WAKE_SOURCE_IRQ25 | IRQ25. |
| LPM_STANDBY_WAKE_SOURCE_IRQ26 | IRQ26. |
| LPM_STANDBY_WAKE_SOURCE_IRQ27 | IRQ27. |
| LPM_STANDBY_WAKE_SOURCE_IRQ28 | IRQ28. |
| LPM_STANDBY_WAKE_SOURCE_IRQ29 | IRQ29. |
| LPM_STANDBY_WAKE_SOURCE_IRQ30 | IRQ30. |
| LPM_STANDBY_WAKE_SOURCE_IRQ31 | IRQ31. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_COMPHS0 | Comparator-HS0 Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0U | ULPT0 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0A | ULPT0 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0B | ULPT0 Compare Match B Interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1U | ULPT1 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1A | ULPT1 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1B | ULPT1 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IRQ16 | IRQ16. |
| LPM_STANDBY_WAKE_SOURCE_IRQ17 | IRQ17. |
| LPM_STANDBY_WAKE_SOURCE_IRQ18 | IRQ18. |
| LPM_STANDBY_WAKE_SOURCE_IRQ19 | IRQ19. |
| LPM_STANDBY_WAKE_SOURCE_IRQ20 | IRQ20. |
| LPM_STANDBY_WAKE_SOURCE_IRQ21 | IRQ21. |
| LPM_STANDBY_WAKE_SOURCE_IRQ22 | IRQ22. |
| LPM_STANDBY_WAKE_SOURCE_IRQ23 | IRQ23. |
| LPM_STANDBY_WAKE_SOURCE_IRQ24 | IRQ24. |
| LPM_STANDBY_WAKE_SOURCE_IRQ25 | IRQ25. |
| LPM_STANDBY_WAKE_SOURCE_IRQ26 | IRQ26. |
| LPM_STANDBY_WAKE_SOURCE_IRQ27 | IRQ27. |
| LPM_STANDBY_WAKE_SOURCE_IRQ28 | IRQ28. |
| LPM_STANDBY_WAKE_SOURCE_IRQ29 | IRQ29. |
| LPM_STANDBY_WAKE_SOURCE_IRQ30 | IRQ30. |
| LPM_STANDBY_WAKE_SOURCE_IRQ31 | IRQ31. |
| LPM_STANDBY_WAKE_SOURCE_IRQ0 | IRQ0. |
| LPM_STANDBY_WAKE_SOURCE_IRQ1 | IRQ1. |
| LPM_STANDBY_WAKE_SOURCE_IRQ2 | IRQ2. |
| LPM_STANDBY_WAKE_SOURCE_IRQ3 | IRQ3. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_IRQ4 | IRQ4. |
| LPM_STANDBY_WAKE_SOURCE_IRQ5 | IRQ5. |
| LPM_STANDBY_WAKE_SOURCE_IRQ6 | IRQ6. |
| LPM_STANDBY_WAKE_SOURCE_IRQ7 | IRQ7. |
| LPM_STANDBY_WAKE_SOURCE_IRQ8 | IRQ8. |
| LPM_STANDBY_WAKE_SOURCE_IRQ9 | IRQ9. |
| LPM_STANDBY_WAKE_SOURCE_IRQ10 | IRQ10. |
| LPM_STANDBY_WAKE_SOURCE_IRQ11 | IRQ11. |
| LPM_STANDBY_WAKE_SOURCE_IRQ12 | IRQ12. |
| LPM_STANDBY_WAKE_SOURCE_IRQ13 | IRQ13. |
| LPM_STANDBY_WAKE_SOURCE_IRQ14 | IRQ14. |
| LPM_STANDBY_WAKE_SOURCE_IRQ15 | IRQ15. |
| LPM_STANDBY_WAKE_SOURCE_IWDT | Independent watchdog interrupt. |
| LPM_STANDBY_WAKE_SOURCE_KEY | Key interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD1 | Low Voltage Detection 1 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_LVD2 | Low Voltage Detection 2 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VBATT | VBATT Monitor interrupt. |
| LPM_STANDBY_WAKE_SOURCE_VRTC | LVDVRTC interrupt. |
| LPM_STANDBY_WAKE_SOURCE_EXLVD | LVDEXLVD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ACMPHS0 | Analog Comparator High-speed 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ACMPLP0 | Analog Comparator Low-speed 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM1 | RTC Alarm interrupt 1. |
| LPM_STANDBY_WAKE_SOURCE_RTCALM | RTC Alarm interrupt. |
| LPM_STANDBY_WAKE_SOURCE_RTCPRD | RTC Period interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_USBHS | USB High-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_USBFS | USB Full-speed interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGTW0UD | AGTW0 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGTW1UD | AGTW1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGTW1CA | AGTW1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGTW1CB | AGTW1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD | AGT1 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CA | AGT1 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1CB | AGT1 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_IIC0 | I2C 0 interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT0UD | AGT0 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT3UD | AGT3 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT1UD_S | AGT1 Underflow interrupt for specific board. |
| LPM_STANDBY_WAKE_SOURCE_AGT3CA | AGT3 Compare Match A interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT2UD | AGT2 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT3CB | AGT3 Compare Match B interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT3UD_S | AGT3 Underflow interrupt for specific board. |
| LPM_STANDBY_WAKE_SOURCE_COMPHS0 | Comparator-HS0 Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT4UD | AGT4 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT5UD | AGT5 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT6UD | AGT6 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_AGT7UD | AGT7 Underflow interrupt. |
| LPM_STANDBY_WAKE_SOURCE_SOSTD | SOSTD interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0U | ULPT0 Underflow Interrupt. |

| | |
|---|---|
| LPM_STANDBY_WAKE_SOURCE_ULP0A | ULPT0 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP0B | ULPT0 Compare Match B Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_I3C0 | I3C0 address match interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1U | ULPT1 Underflow Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1A | ULPT1 Compare Match A Interrupt. |
| LPM_STANDBY_WAKE_SOURCE_ULP1B | ULPT1 Compare Match B Interrupt. |

### ◆ lpm_io_port_t

| enum lpm_io_port_t | |
|---|---|
| I/O port state after Deep Software Standby mode | |
| Enumerator | |
| LPM_IO_PORT_RESET | When the Deep Software Standby mode is canceled, the I/O ports are in the reset state |
| LPM_IO_PORT_NO_CHANGE | When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode |

### ◆ lpm_power_supply_t

| enum lpm_power_supply_t | |
|---|---|
| Power supply control | |
| Enumerator | |
| LPM_POWER_SUPPLY_DEEPCUT0 | Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode |
| LPM_POWER_SUPPLY_DEEPCUT1 | Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode |
| LPM_POWER_SUPPLY_DEEPCUT3 | Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled |
| LPM_POWER_SUPPLY_DEEP_STANDBY_MODE1 | Power to the standby RAM, Low-speed on-chip oscillator, Programmable Voltage Detection Unit 0, and USBFS/HS resume detecting unit is supplied in deep software standby mode. |
| LPM_POWER_SUPPLY_DEEP_STANDBY_MODE2 | Power to standby RAM, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode. Power to the Programmable Voltage Detection Unit 0 is supplied in deep software standby mode. |
| LPM_POWER_SUPPLY_DEEP_STANDBY_MODE3 | Power to standby RAM, Programmable Voltage Detection Unit 0, USBFS/HS resume detecting unit, Low-speed on-chip oscillator, and IWDT is disabled in deep software standby mode. |

### ◆ lpm_deep_standby_cancel_edge_t

| enum lpm_deep_standby_cancel_edge_t | |
|---|---|
| Deep Standby Interrupt Edge | |
| Enumerator | |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE | No options for a deep standby cancel source. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING | IRQ0-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING | IRQ0-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING | IRQ1-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING | IRQ1-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING | IRQ2-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING | IRQ2-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING | IRQ3-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING | IRQ3-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING | IRQ4-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING | IRQ4-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING | IRQ5-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING | IRQ5-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING | IRQ6-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING | IRQ6-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING | IRQ7-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING | IRQ7-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING | IRQ8-DS Pin Rising Edge. |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING | IRQ8-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING | IRQ9-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING | IRQ9-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING | IRQ10-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING | IRQ10-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING | IRQ11-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING | IRQ11-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING | IRQ12-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING | IRQ12-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING | IRQ13-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING | IRQ13-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING | IRQ14-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING | IRQ14-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING | IRQ14-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING | IRQ14-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING | LVD1 Rising Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING | LVD1 Falling Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING | LVD2 Rising Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING | LVD2 Falling Slope. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING | NMI Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING | NMI Pin Falling Edge. |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16_RISING | IRQ16-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16_FALLING | IRQ16-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17_RISING | IRQ17-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17_FALLING | IRQ17-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18_RISING | IRQ18-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18_FALLING | IRQ18-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19_RISING | IRQ19-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19_FALLING | IRQ19-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20_RISING | IRQ20-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20_FALLING | IRQ20-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21_RISING | IRQ21-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21_FALLING | IRQ21-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22_RISING | IRQ22-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22_FALLING | IRQ22-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23_RISING | IRQ23-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23_FALLING | IRQ23-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24_RISING | IRQ24-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24_FALLING | IRQ24-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25_RISING | IRQ25-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25_FALLING | IRQ25-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26_RISING | IRQ26-DS Pin Rising Edge. |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26_FALLING | IRQ26-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27_RISING | IRQ27-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27_FALLING | IRQ27-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28_RISING | IRQ28-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28_FALLING | IRQ28-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29_RISING | IRQ29-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29_FALLING | IRQ29-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30_RISING | IRQ30-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30_FALLING | IRQ30-DS Pin Falling Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31_RISING | IRQ31-DS Pin Rising Edge. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31_FALLING | IRQ31-DS Pin Falling Edge. |

### ◆ lpm_deep_standby_soft_start_mode_t

| enum lpm_deep_standby_soft_start_mode_t | |
|---|---|
| Select soft start mode in deep software standby mode. This setting configures the wake time and inrush current when waking from deep software standby mode (See the device user manual for more details). | |
| Enumerator | |
| LPM_DEEP_STANDBY_SOFT_START_MODE_0 | Soft Start Mode 0 (DCSSMODE=0) |
| LPM_DEEP_STANDBY_SOFT_START_MODE_1 | Soft Start Mode 1 (DCSSMODE=1) |
| LPM_DEEP_STANDBY_SOFT_START_MODE_2 | Soft Start Mode 2 (DCSSMODE=2) |
| LPM_DEEP_STANDBY_SOFT_START_MODE_3 | Soft Start Mode 3 (DCSSMODE=3) |

### ◆ lpm_deep_standby_cancel_source_t

| enum lpm_deep_standby_cancel_source_t |
|---|

| Deep Standby cancel sources | |
|---|---|
| Enumerator | |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY | Cancel deep standby only by reset. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0 | IRQ0. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1 | IRQ1. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2 | IRQ2. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3 | IRQ3. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4 | IRQ4. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5 | IRQ5. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6 | IRQ6. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7 | IRQ7. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8 | IRQ8. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9 | IRQ9. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10 | IRQ10. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11 | IRQ11. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12 | IRQ12. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13 | IRQ13. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14 | IRQ14. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15 | IRQ15. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1 | LVD1. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2 | LVD2. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL | RTC Interval Interrupt. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM | RTC Alarm Interrupt. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI | NMI. |
| | |

| | |
|---|---|
| LPM_DEEP_STANDBY_CANCEL_SOURCE_USBFS | USBFS Suspend/Resume. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_USBHS | USBHS Suspend/Resume. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT1 | AGT1 Underflow. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT3 | AGT3 Underflow. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_ULPT0 | ULPT0 Overflow. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_ULPT1 | ULPT1 Overflow. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IWDT | IWDT Underflow. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_VBATT | VBATT Tamper Detection. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ16 | IRQ16. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ17 | IRQ17. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ18 | IRQ18. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ19 | IRQ19. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ20 | IRQ20. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ21 | IRQ21. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ22 | IRQ22. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ23 | IRQ23. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ24 | IRQ24. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ25 | IRQ25. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ26 | IRQ26. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ27 | IRQ27. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ28 | IRQ28. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ29 | IRQ29. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ30 | IRQ30. |
| LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ31 | IRQ31. |

### ◆ lpm_output_port_enable_t

| enum lpm_output_port_enable_t | |
|---|---|
| Output port enable | |
| Enumerator | |
| LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE | 0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode. |
| LPM_OUTPUT_PORT_ENABLE_RETAIN | 1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state. |

### ◆ lpm_ldo_standby_operation_t

| enum lpm_ldo_standby_operation_t | |
|---|---|
| Configure the behavior of an oscillator's LDO in standby mode. | |
| Enumerator | |
| LPM_LDO_STANDBY_OPERATION_DISABLED | The LDO is disabled in standby mode. |
| LPM_LDO_STANDBY_OPERATION_RETAINED | The LDO state is retained during standby mode. |

### ◆ lpm_flash_mode_select_t

| enum lpm_flash_mode_select_t | |
|---|---|
| Flash operating mode select. | |
| Enumerator | |
| LPM_FLASH_MODE_SELECT_ACTIVE | Flash active. |
| LPM_FLASH_MODE_SELECT_STOP | Flash stop. |

### ◆ lpm_hoco_startup_speed_t

| enum lpm_hoco_startup_speed_t | |
|---|---|
| Starting the high-speed on-chip oscillator at the times of release from SSTBY mode and of transitions to SNOOZE mode. | |
| Enumerator | |
| LPM_HOCO_STARTUP_SPEED_NORMAL_SPEED | Starting of the high-speed on-chip oscillator is at normal speed. |
| LPM_HOCO_STARTUP_SPEED_HIGH_SPEED | Starting of the high-speed on-chip oscillator is at high speed. |

### ◆ lpm_standby_sosc_t

| enum lpm_standby_sosc_t | |
|---|---|
| SOSC setting in SSTBY mode or in SNOOZE mode. | |
| Enumerator | |
| LPM_STANDBY_SOSC_ENABLE | Enables supply of SOSC clock to peripheral functions. |
| LPM_STANDBY_SOSC_DISABLE | Stops supply SOSC clock to peripheral functions other than the Realtime clock. |

## 5.3.13 Security

Interfaces

### Detailed Description

Security Interfaces.

### Modules

| | |
|---|---|
| | RSIP Interface<br>Interface for Renesas Secure IP (RSIP) functions. |
| | RSIP key injection Interface<br>Interface for key injection by Renesas Secure IP (RSIP) functions. |

| | |
|---|---|
| | SCE Interface<br>Interface for Secure Crypto Engine (SCE) functions. |
| | SCE key injection Interface<br>Interface for key injection by Secure Crypto Engine (SCE) functions. |

### 5.3.13.1 RSIP Interface

Interfaces » Security

#### Detailed Description

Interface for Renesas Secure IP (RSIP) functions.

## Summary

The RSIP interface provides RSIP functionality.

#### Data Structures

| | |
|---|---|
| struct | rsip_wrapped_key_t |
| struct | rsip_wrapped_dkm_t |
| struct | rsip_cfg_t |
| struct | rsip_api_t |
| struct | rsip_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | rsip_sha_handle_abstractor_t |
| typedef void | rsip_hmac_handle_abstractor_t |
| typedef void | rsip_kdf_sha_handle_abstractor_t |
| typedef void | rsip_kdf_hmac_handle_abstractor_t |
| typedef void | rsip_wrapped_secret_abstractor_t |
| typedef void | rsip_verified_cert_info_abstractor_t |

| | |
|---|---|
| typedef void | rsip_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rsip_key_type_t |
| enum | rsip_aes_cipher_mode_t |
| enum | rsip_aes_aead_mode_t |
| enum | rsip_chacha_poly_mode_t |
| enum | rsip_aes_mac_mode_t |
| enum | rsip_hash_type_t |
| enum | rsip_mgf_type_t |
| enum | rsip_rsa_salt_length_t |
| enum | rsip_initial_vector_type_t |
| enum | rsip_otf_channel_t |

## Data Structure Documentation

### ◆ rsip_wrapped_key_t

| struct rsip_wrapped_key_t | | |
|---|---|---|
| Wrapped key structure for all supported algorithms. | | |
| Data Fields | | |
| rsip_key_type_t | type | Key type. |
| void * | p_value | Key value. |

### ◆ rsip_wrapped_dkm_t

| struct rsip_wrapped_dkm_t |
|---|
| Wrapped Derived Keying Material (DKM) structure for KDF APIs. |

### ◆ rsip_cfg_t

| struct rsip_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| void const * | p_extend | Hardware-dependent configuration. |

### ◆ rsip_api_t

| struct rsip_api_t | |
|---|---|
| RSIP driver structure. General RSIP functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rsip_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | randomNumberGenerate )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_random) |
| | |
| fsp_err_t(* | keyGenerate )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | keyPairGenerate )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key) |
| | |
| fsp_err_t(* | encryptedKeyWrap )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | rfc3394_KeyWrap )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key) |
| | |
| fsp_err_t(* | rfc3394_KeyUnwrap )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key) |
| | |
| fsp_err_t(* | publicKeyExport )(rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t *const p_raw_public_key) |
| | |
| fsp_err_t(* | aesCipherInit )(rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector) |
| | |

| | |
|---|---|
| fsp_err_t(* | aesCipherUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length) |
| | |
| fsp_err_t(* | aesCipherFinish )(rsip_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | aesAeadInit )(rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length) |
| | |
| fsp_err_t(* | aesAeadLengthsSet )(rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length) |
| | |
| fsp_err_t(* | aesAeadAadUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length) |
| | |
| fsp_err_t(* | aesAeadUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| | |
| fsp_err_t(* | aesAeadFinish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag) |
| | |
| fsp_err_t(* | aesAeadVerify )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length) |
| | |
| fsp_err_t(* | aesMacInit )(rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key) |
| | |
| fsp_err_t(* | aesMacUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| | |
| fsp_err_t(* | aesMacSignFinish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac) |
| | |
| fsp_err_t(* | aesMacVerifyFinish )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length) |
| | |
| fsp_err_t(* | chacha20Init )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const |

| | |
|---|---|
| | *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter) |
| | |
| fsp_err_t(* | chacha20Update )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| | |
| fsp_err_t(* | chacha20Finish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length) |
| | |
| fsp_err_t(* | chacha20Poly1305Init )(rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length) |
| | |
| fsp_err_t(* | chacha20Poly1305AadUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length) |
| | |
| fsp_err_t(* | chacha20Poly1305Update )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length) |
| | |
| fsp_err_t(* | chacha20Poly1305Finish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag) |
| | |
| fsp_err_t(* | chacha20Poly1305Verify )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length) |
| | |
| fsp_err_t(* | ecdsaSign )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature) |
| | |
| fsp_err_t(* | ecdsaVerify )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| | |
| fsp_err_t(* | eddsaSign )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature) |
| | |

| | |
|---|---|
| fsp_err_t(* | eddsaVerify )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature) |
| | |
| fsp_err_t(* | ecdhKeyAgree )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret) |
| | |
| fsp_err_t(* | ecdhPlainKeyAgree )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret) |
| | |
| fsp_err_t(* | rsaEncrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher) |
| | |
| fsp_err_t(* | rsaDecrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain) |
| | |
| fsp_err_t(* | rsaesPkcs1V15Encrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher) |
| | |
| fsp_err_t(* | rsaesPkcs1V15Decrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length) |
| | |
| fsp_err_t(* | rsaesOaepEncrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher) |
| | |
| fsp_err_t(* | rsaesOaepDecrypt )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length) |

| | |
|---|---|
| | |
| fsp_err_t(* | rsassaPkcs1V15Sign )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature) |
| | |
| fsp_err_t(* | rsassaPkcs1V15Verify )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| | |
| fsp_err_t(* | rsassaPssSign )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature) |
| | |
| fsp_err_t(* | rsassaPssVerify )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| | |
| fsp_err_t(* | shaCompute )(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest) |
| | |
| fsp_err_t(* | shaInit )(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type) |
| | |
| fsp_err_t(* | shaUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| | |
| fsp_err_t(* | shaFinish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest) |
| | |
| fsp_err_t(* | shaSuspend )(rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t *const p_handle) |
| | |
| fsp_err_t(* | shaResume )(rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t const *const p_handle) |
| | |
| fsp_err_t(* | hmacCompute )(rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t |

| | |
|---|---|
| | *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac) |
| | |
| fsp_err_t(* | hmacVerify )(rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length) |
| | |
| fsp_err_t(* | hmacInit )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key) |
| | |
| fsp_err_t(* | hmacUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| | |
| fsp_err_t(* | hmacSignFinish )(rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac) |
| | |
| fsp_err_t(* | hmacVerifyFinish )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length) |
| | |
| fsp_err_t(* | hmacSuspend )(rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t *const p_handle) |
| | |
| fsp_err_t(* | hmacResume )(rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t const *const p_handle) |
| | |
| fsp_err_t(* | pkiEcdsaCertVerify )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature) |
| | |
| fsp_err_t(* | pkiVerifiedCertInfoExport )(rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t *const p_verified_cert_info) |
| | |
| fsp_err_t(* | pkiVerifiedCertInfoImport )(rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t const *const p_verified_cert_info) |
| | |
| fsp_err_t(* | pkiCertKeyImport )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key) |

| | |
|---|---|
| | |
| fsp_err_t(* | kdfShaInit )(rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type) |
| | |
| fsp_err_t(* | kdfShaEcdhSecretUpdate )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret) |
| | |
| fsp_err_t(* | kdfShaUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| | |
| fsp_err_t(* | kdfShaFinish )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm) |
| | |
| fsp_err_t(* | kdfShaSuspend )(rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t *const p_handle) |
| | |
| fsp_err_t(* | kdfShaResume )(rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t const *const p_handle) |
| | |
| fsp_err_t(* | kdfHmacDkmKeyImport )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | kdfHmacEcdhSecretKeyImport )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | kdfHmacInit )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key) |
| | |
| fsp_err_t(* | kdfHmacDkmUpdate )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm) |
| | |
| fsp_err_t(* | kdfHmacEcdhSecretUpdate )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret) |
| | |
| fsp_err_t(* | kdfHmacUpdate )(rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length) |
| | |

| | |
|---|---|
| fsp_err_t(* | kdfHmacSignFinish )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm) |
| | |
| fsp_err_t(* | kdfHmacSuspend )(rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t *const p_handle) |
| | |
| fsp_err_t(* | kdfHmacResume )(rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t const *const p_handle) |
| | |
| fsp_err_t(* | kdfDkmConcatenate )(rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length) |
| | |
| fsp_err_t(* | kdfDerivedKeyImport )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | kdfDerivedIvWrap )(rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector) |
| | |
| fsp_err_t(* | otfInit )(rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rsip_api_t::open) (rsip_ctrl_t *const p_ctrl, rsip_cfg_t const *const p_cfg)

Enables use of Renesas Secure IP functionality.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rsip_api_t::close) (rsip_ctrl_t *const p_ctrl)

Disables use of Renesas Secure IP functionality.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |

### ◆ randomNumberGenerate

fsp_err_t(* rsip_api_t::randomNumberGenerate) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_random)

Generates a 128-bit random number.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_random | 128bit random numbers. |

### ◆ keyGenerate

fsp_err_t(* rsip_api_t::keyGenerate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_key)

Generate a wrapped symmetric key from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in,out] | p_wrapped_key | Pointer to destination of wrapped key. The length depends on key type. Refer "Key Size Table". |

### ◆ keyPairGenerate

fsp_err_t(* rsip_api_t::keyPairGenerate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t *const p_wrapped_public_key, rsip_wrapped_key_t *const p_wrapped_private_key)

Generate a wrapped asymmetric key pair from a random number. In this API, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in,out] | p_wrapped_public_key | Key index for Public Key. The length depends on the key type. Refer "Key Size Table". |
| [in,out] | p_wrapped_private_key | Key index for Private Key. The length depends on the key type. Refer "Key Size Table". |

### ◆ encryptedKeyWrap

fsp_err_t(* rsip_api_t::encryptedKeyWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_key_update_key, void const *const p_initial_vector, void const *const p_encrypted_key, rsip_wrapped_key_t *const p_wrapped_key)

Decrypt the encrypted user key with Key Update Key (KUK) and wrap it with the Hardware Unique Key (HUK).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_key_update_key | Pointer to Key Update Key. |
| [in] | p_initial_vector | Initialization vector when generating encrypted_key. The length is 16 bytes. |
| [in] | p_encrypted_key | Encrypted user key. The length depends on the key type. Refer "Key Size Table". |
| [in,out] | p_wrapped_key | Pointer to destination of wrapped key. The length depends on key type. Refer "Key Size Table". |

### ◆ rfc3394_KeyWrap

fsp_err_t(* rsip_api_t::rfc3394_KeyWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, rsip_wrapped_key_t const *const p_wrapped_target_key, uint8_t *const p_rfc3394_wrapped_target_key)

This function provides Key Wrap algorithm compliant with RFC3394. Using p_wrapped_kek to wrap p_wrapped_target_key, and output the result to p_rfc3394_wrapped_target_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_kek | Pointer to wrapped key-encryption-key used to RFC3394-wrap the target key. |
| [in] | p_wrapped_target_key | Pointer to wrapped target key to be RFC3394-wrapped. |
| [out] | p_rfc3394_wrapped_target_key | Pointer to destination of RFC3394-wrapped target key. |

### ◆ rfc3394_KeyUnwrap

fsp_err_t(* rsip_api_t::rfc3394_KeyUnwrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_kek, uint8_t const *const p_rfc3394_wrapped_target_key, rsip_wrapped_key_t *const p_wrapped_target_key)

This function provides Key Unwrap algorithm compliant with RFC3394. Using p_wrapped_kek to unwrap p_rfc3394_wrapped_target_key, and output the result to p_wrapped_target_key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_kek | Pointer to wrapped key-encryption-key used to RFC3394-unwrap the target key. |
| [in] | p_rfc3394_wrapped_target_key | Pointer to AES-wrapped target key to be RFC3394-unwrapped. |
| [in,out] | p_wrapped_target_key | Pointer to destination of RFC3394-unwrapped target key. |

### ◆ publicKeyExport

fsp_err_t(* rsip_api_t::publicKeyExport) (rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t *const p_raw_public_key)

Exports public key parameters from a wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | p_wrapped_public_key | Key index for Public Key. The length depends on the key type. Refer "Key Size Table". |
| [out] | p_raw_public_key | Pointer to destination of raw public key. The length depends on the key length. |

### ◆ aesCipherInit

fsp_err_t(* rsip_api_t::aesCipherInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_cipher_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_initial_vector)

Set parameters of AES cipher.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | mode | Block cipher modes of operation for AES. |
| [in] | p_wrapped_key | Pointer to wrapped key of AES or XTS-AES key. |
| [in] | p_initial_vector | Pointer to initialization vector (IV) or nonce. The length is 16 bytes.<br>• [ECB] Not required<br>• [CBC][XTS] IV<br>• [CTR] Nonce |

### ◆ aesCipherUpdate

fsp_err_t(* rsip_api_t::aesCipherUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint8_t *const p_output, uint32_t const length)

Encrypt plaintext.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_input | Pointer to input text. The length is given as the argument. |
| [out] | p_output | Pointer to destination of output text. The length is given as the argument. |
| [in] | length | Byte length of input and output.<br>• [ECB][CBC][CTR] Must be 0 or a multiple of 16.<br>• [XTS] Must be 0 or greater than or equal to 16. After an integer not divisible by 16 is input, update can no longer be executed. |

### ◆ aesCipherFinish

fsp_err_t(* rsip_api_t::aesCipherFinish) (rsip_ctrl_t *const p_ctrl)

Finalize AES operation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |

### ◆ aesAeadInit

fsp_err_t(* rsip_api_t::aesAeadInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_aead_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)

Prepares an AES-AEAD function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | mode | AEAD mode of operation. |
| [in] | p_wrapped_key | Pointer to wrapped key of AES key. |
| [in] | p_nonce | Pointer to nonce. The length is nonce_length. |
| [in] | nonce_length | Byte length of nonce. Input 1 or more. |

### ◆ aesAeadLengthsSet

fsp_err_t(* rsip_api_t::aesAeadLengthsSet) (rsip_ctrl_t *const p_ctrl, uint32_t const total_aad_length, uint32_t const total_text_length, uint32_t const tag_length)

Set text and tag lengths for specific mode.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | total_aad_length | Total AAD length. |
| [in] | total_text_length | Total input and output text length. |
| [in] | tag_length | Input or output tag length. |

### ◆ aesAeadAadUpdate

fsp_err_t(* rsip_api_t::aesAeadAadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)

Inputs additional authentication data.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_aad | Additional authentication data. The length depends on aad_length. |
| [in] | aad_length | Byte length of additional authentication data (0 or more bytes). After starting input of plaintext, this value must always be 0. |

### ◆ aesAeadUpdate

fsp_err_t(* rsip_api_t::aesAeadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Inputs test and executes encryption and decryption.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_input | Pointer to input text. The length is input_length. |
| [in] | input_length | Byte length of input text (0 or more bytes). |
| [out] | p_output | Pointer to destination of output text. The length is p_output_length. |
| [out] | p_output_length | Pointer to destination of output text length. |

### ◆ aesAeadFinish

fsp_err_t(* rsip_api_t::aesAeadFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)

Finalizes an AES-GCM encryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_plain that was input by R_RSIP_AES_GCM_EncryptUpdate(), this API will output the result of encrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_output | Pointer to destination of output text. The fractional block is output. |
| [out] | p_output_length | Pointer to destination of output text length. |
| [out] | p_tag | Pointer to destination of tag for authentication. GCM : The length is 16 bytes. *If a different tag length is required, truncate the 16-byte tag to the required tag length (NIST SP800-38D 7.1). CCM : The length is the value set by the API R_RSIP_AES_AEAD_LengthsSet(). |

### ◆ aesAeadVerify

fsp_err_t(* rsip_api_t::aesAeadVerify) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)

Finalizes an AES-GCM decryption.

If there is 16-byte fractional data indicated by the total data length of the value of p_cipher that was input by R_RSIP_AES_GCM_DecryptUpdate(), this API will output the result of decrypting that fractional data to p_cipher. Here, the portion that does not reach 16 bytes will be padded with zeros.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_output | Pointer to destination of decrypted data. |
| [out] | p_output_length | Pointer to destination of decrypted data length. |
| [in] | p_tag | Pointer to destination of tag for authentication. The length depends on tag_length. |
| [in] | tag_length | Byte length of tag. Must be 1 to 16. |

### ◆ aesMacInit

fsp_err_t(* rsip_api_t::aesMacInit) (rsip_ctrl_t *const p_ctrl, rsip_aes_mac_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares an AES-MAC generation and verification.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | mode | MAC mode of operation |
| [in] | p_wrapped_key | Pointer to wrapped key of AES key. |

### ◆ aesMacUpdate

fsp_err_t(* rsip_api_t::aesMacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Input message. Inside this function, the data that is input by the user is buffered until the input value of p_message exceeds 16 bytes. If the input value, p_message, is not a multiple of 16 bytes, it will be padded within the function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |

### ◆ aesMacSignFinish

fsp_err_t(* rsip_api_t::aesMacSignFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)

Finalizes an AES-CMAC generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_mac | Pointer to destination of MAC. The length is 16 bytes. |

### ◆ aesMacVerifyFinish

fsp_err_t(* rsip_api_t::aesMacVerifyFinish) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)

Finalizes an AES-CMAC verification.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_mac | Pointer to MAC. The length depends on mac_length. |
| [in] | mac_length | Byte length of MAC. Must be 2 to 16. |

### ◆ chacha20Init

fsp_err_t(* rsip_api_t::chacha20Init) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const counter)

Prepares a ChaCha20 function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_key | Pointer to wrapped key of ChaCha20 key. |
| [in] | p_nonce | Pointer to nonce. The length is 12 bytes. |
| [in] | counter | 32-bit initial counter. This is usually zero. |

### ◆ chacha20Update

fsp_err_t(* rsip_api_t::chacha20Update) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Encrypts plaintext or decrypts ciphertext.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_input | Pointer to input text. The length is input_length. |
| [in] | input_length | Byte length of input text (0 or more bytes). |
| [out] | p_output | Pointer to destination of output text. The length is p_output_length. |
| [out] | p_output_length | Pointer to destination of output text length. |

### ◆ chacha20Finish

fsp_err_t(* rsip_api_t::chacha20Finish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length)

Finalize ChaCha20 operation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_output | Pointer to destination of output text. The fractional block is output. |
| [out] | p_output_length | Pointer to destination of output text length. |

### ◆ chacha20Poly1305Init

fsp_err_t(* rsip_api_t::chacha20Poly1305Init) (rsip_ctrl_t *const p_ctrl, rsip_chacha_poly_mode_t const mode, rsip_wrapped_key_t const *const p_wrapped_key, uint8_t const *const p_nonce, uint32_t const nonce_length)

Prepares a ChaCha20-Poly1305 function.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | mode | AEAD mode of operation. |
| [in] | p_wrapped_key | Pointer to wrapped key of AES key. |
| [in] | p_nonce | Pointer to nonce. The length is nonce_length. |
| [in] | nonce_length | Byte length of nonce. Input 1 or more. |

### ◆ chacha20Poly1305AadUpdate

fsp_err_t(* rsip_api_t::chacha20Poly1305AadUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_aad, uint32_t const aad_length)

Inputs Additional Authentication Data (AAD).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_aad | Additional authentication data. The length depends on aad_length. |
| [in] | aad_length | Byte length of additional authentication data (0 or more bytes). After starting input of plaintext, this value must always be 0. |

### ◆ chacha20Poly1305Update

fsp_err_t(* rsip_api_t::chacha20Poly1305Update) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_input, uint32_t const input_length, uint8_t *const p_output, uint32_t *const p_output_length)

Encrypts plaintext or decrypts ciphertext.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_input | Pointer to input text. The length is input_length. |
| [in] | input_length | Byte length of input text (0 or more bytes). |
| [out] | p_output | Pointer to destination of output text. The length is p_output_length. |
| [out] | p_output_length | Pointer to destination of output text length. |

### ◆ chacha20Poly1305Finish

fsp_err_t(* rsip_api_t::chacha20Poly1305Finish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t *const p_tag)

Finalizes a ChaCha20-Poly1305 encryption.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_output | Pointer to destination of output text. The fractional block is output. |
| [out] | p_output_length | Pointer to destination of output text length. |
| [out] | p_tag | Pointer to destination of tag for authentication. The length is 16 bytes. |

### ◆ chacha20Poly1305Verify

fsp_err_t(* rsip_api_t::chacha20Poly1305Verify) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_output, uint32_t *const p_output_length, uint8_t const *const p_tag, uint32_t const tag_length)

Finalizes a ChaCha20-Poly1305 decryption.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_output | Pointer to destination of decrypted data. |
| [out] | p_output_length | Pointer to destination of decrypted data length. |
| [in] | p_tag | Pointer to destination of tag for authentication. The length depends on tag_length. |
| [in] | tag_length | Byte length of tag. Must be 16. |

### ◆ ecdsaSign

fsp_err_t(* rsip_api_t::ecdsaSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs a hashed message. The message hash should be generated in advance.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of ECC private key. |
| [in] | p_hash | Pointer to hash value. The length is as same as the key length. |
| [out] | p_signature | Pointer to destination of signature (r, s). The length is twice as long as the key length. |

### ◆ ecdsaVerify

fsp_err_t(* rsip_api_t::ecdsaVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies a hashed message. The message hash should be generated in advance.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of ECC public key. |
| [in] | p_hash | Pointer to hash value. The length is as same as the key length. |
| [in] | p_signature | Pointer to signature (r, s). The length is twice as long as the key length. |

### ◆ eddsaSign

fsp_err_t(* rsip_api_t::eddsaSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t *const p_signature)

Sign the message using EdDSA.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of ECC private key. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of ECC public key. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message. |
| [out] | p_signature | Pointer to destination of signature (r, s). The length is twice as long as the key length. |

### ◆ eddsaVerify

fsp_err_t(* rsip_api_t::eddsaVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_message, uint64_t const message_length, uint8_t const *const p_signature)

Verifies the message using EdDSA.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of ECC public key. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message. |
| [in] | p_signature | Pointer to signature (r, s). The length is twice as long as the key length. |

### ◆ ecdhKeyAgree

fsp_err_t(* rsip_api_t::ecdhKeyAgree) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)

Computes ECDH secret with wrapped private key and wrapped public key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Wrapped secp256r1 private key. |
| [in] | p_wrapped_public_key | Wrapped secp256r1 public key. |
| [out] | p_wrapped_secret | Pointer to destination of wrapped secret |

### ◆ ecdhPlainKeyAgree

fsp_err_t(* rsip_api_t::ecdhPlainKeyAgree) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_plain_public_key, rsip_wrapped_secret_abstractor_t *const p_wrapped_secret)

Computes ECDH secret with wrapped private key and plain public key.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Wrapped secp256r1 private key. |
| [in] | p_plain_public_key | Plain secp256r1 public key. |
| [out] | p_wrapped_secret | Pointer to destination of wrapped secret. |

### ◆ rsaEncrypt

fsp_err_t(* rsip_api_t::rsaEncrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint8_t *const p_cipher)

Encrypts plaintext with raw RSA.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | p_plain | Pointer to plaintext. The length is as same as the key length. |
| [out] | p_cipher | Pointer to destination of ciphertext. The length is as same as the key length. |

### ◆ rsaDecrypt

fsp_err_t(* rsip_api_t::rsaDecrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain)

Decrypts ciphertext with raw RSA.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of RSA private key. |
| [in] | p_cipher | Pointer to ciphertext. The length is as same as the key length. |
| [out] | p_plain | Pointer to destination of plaintext. The length is as same as the key length. |

### ◆ rsaesPkcs1V15Encrypt

fsp_err_t(* rsip_api_t::rsaesPkcs1V15Encrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)

Encrypts plaintext with RSAES-PKCS1-v1_5.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | p_plain | Pointer to plaintext. |
| [in] | plain_length | Length of plaintext. |
| [out] | p_cipher | Pointer to destination of ciphertext. The length is as same as the key length. |

### ◆ rsaesPkcs1V15Decrypt

fsp_err_t(* rsip_api_t::rsaesPkcs1V15Decrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)

Decrypts with RSAES-PKCS1-v1_5.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of RSA private key. |
| [in] | p_cipher | Pointer to ciphertext. The length is as same as the key length. |
| [out] | p_plain | Pointer to destination of plaintext. |
| [out] | p_plain_length | Pointer to destination of actual plaintext length. |
| [in] | plain_buffer_length | Length of plaintext destination. It must be equal to or greater than *p_plain_length. |

### ◆ rsaesOaepEncrypt

fsp_err_t(* rsip_api_t::rsaesOaepEncrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_plain, uint32_t const plain_length, uint8_t *const p_cipher)

Encrypts plaintext with RSAES-OAEP.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | hash_function | Hash function for label. |
| [in] | mask_generation_function | Mask generation function in EME-OAEP encoding. |
| [in] | p_label | Pointer to label. If label_length != 0, p_label must not be NULL. |
| [in] | label_length | Length of label. Please set 0 or more. |
| [in] | p_plain | Pointer to plaintext. |
| [in] | plain_length | Length of plaintext. |
| [out] | p_cipher | Pointer to destination of ciphertext. The length is as same as the key length. |

### ◆ rsaesOaepDecrypt

fsp_err_t(* rsip_api_t::rsaesOaepDecrypt) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint8_t const *const p_label, uint32_t const label_length, uint8_t const *const p_cipher, uint8_t *const p_plain, uint32_t *const p_plain_length, uint32_t const plain_buffer_length)

Decrypts ciphertext with RSAES-OAEP.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of RSA private key. |
| [in] | hash_function | Hash function for label. |
| [in] | mask_generation_function | Mask generation function in EME-OAEP encoding. |
| [in] | p_label | Pointer to label. If label_length != 0, p_label must not be NULL. |
| [in] | label_length | Length of label. Please set 0 or more. |
| [in] | p_cipher | Pointer to ciphertext. The length is as same as the key length. |
| [out] | p_plain | Pointer to destination of plaintext. |
| [out] | p_plain_length | Pointer to destination of actual plaintext length. |
| [in] | plain_buffer_length | Length of plaintext destination. It must be equal to or greater than *p_plain_length. |

### ◆ rsassaPkcs1V15Sign

fsp_err_t(* rsip_api_t::rsassaPkcs1V15Sign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs message with RSASSA-PKCS1-v1_5.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of RSA private key. |
| [in] | hash_function | Hash function in EMSA-PKCS1-v1_5. |
| [in] | p_hash | Pointer to input hash. |
| [out] | p_signature | Pointer to destination of signature. |

### ◆ rsassaPkcs1V15Verify

fsp_err_t(* rsip_api_t::rsassaPkcs1V15Verify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies signature with RSASSA-PKCS1-v1_5.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | hash_function | Hash function in EMSA-PKCS1-v1_5. |
| [in] | p_hash | Pointer to input hash. |
| [in] | p_signature | Pointer to input signature. |

### ◆ rsassaPssSign

fsp_err_t(* rsip_api_t::rsassaPssSign) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_private_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t *const p_signature)

Signs message with RSASSA-PSS.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_private_key | Pointer to wrapped key of RSA private key. |
| [in] | hash_function | Hash function in EMSA-PSS-ENCODE. |
| [in] | mask_generation_function | Mask generation function in EMSA-PSS-ENCODE. |
| [in] | salt_length | Salt length. |
| [in] | p_hash | Pointer to input hash. |
| [out] | p_signature | Pointer to destination of signature. |

### ◆ rsassaPssVerify

fsp_err_t(* rsip_api_t::rsassaPssVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, rsip_hash_type_t const hash_function, rsip_mgf_type_t const mask_generation_function, uint32_t const salt_length, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies signature with RSASSA-PSS.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of RSA public key. |
| [in] | hash_function | Hash function in EMSA-PSS-VERIFY. |
| [in] | mask_generation_function | Mask generation function in EMSA-PSS-VERIFY. |
| [in] | salt_length | Salt length. |
| [in] | p_hash | Pointer to input hash. |
| [in] | p_signature | Pointer to input signature. |

### ◆ shaCompute

fsp_err_t(* rsip_api_t::shaCompute) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_digest)

Generates SHA message digest.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | hash_type | Generating hash type. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |
| [out] | p_digest | Pointer to destination of message digest. The length depends on hash type. |

### ◆ shaInit

fsp_err_t(* rsip_api_t::shaInit) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)

Prepares a SHA generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | hash_type | Generating hash type. |

### ◆ shaUpdate

fsp_err_t(* rsip_api_t::shaUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |

### ◆ shaFinish

fsp_err_t(* rsip_api_t::shaFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_digest)

Finalizes a SHA generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_digest | Pointer to destination of message digest. The length depends on hash type. |

### ◆ shaSuspend

fsp_err_t(* rsip_api_t::shaSuspend) (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t *const p_handle)

Suspend SHA generation. This API allows you to suspend processing, for example, if you are in the middle of computing digest value for successive chunks of the message and need to perform another process.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_handle | Pointer to destination of SHA control block. |

### ◆ shaResume

fsp_err_t(* rsip_api_t::shaResume) (rsip_ctrl_t *const p_ctrl, rsip_sha_handle_abstractor_t const *const p_handle)

Resume SHA generation. This API allows you to resume a process that has been suspended by R_RSIP_SHA_Suspend() API.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_handle | Pointer to SHA control block. |

### ◆ hmacCompute

fsp_err_t(* rsip_api_t::hmacCompute) (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t *const p_mac)

Generates HMAC.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_key | Pointer to wrapped key of HMAC key. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |
| [out] | p_mac | Pointer to destination of message digest. The length depends on MAC type. |

### ◆ hmacVerify

fsp_err_t(* rsip_api_t::hmacVerify) (rsip_ctrl_t *const p_ctrl, const rsip_wrapped_key_t *p_wrapped_key, uint8_t const *const p_message, uint32_t const message_length, uint8_t const *const p_mac, uint32_t const mac_length)

Verifies HMAC.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_key | Pointer to wrapped key of HMAC key. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |
| [in] | p_mac | Pointer to MAC. The length depends on mac_length. |
| [in] | mac_length | Byte length of MAC. |

### ◆ hmacInit

fsp_err_t(* rsip_api_t::hmacInit) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares a HMAC generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_key | Pointer to wrapped key of HMAC key. |

### ◆ hmacUpdate

fsp_err_t(* rsip_api_t::hmacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |

### ◆ hmacSignFinish

fsp_err_t(* rsip_api_t::hmacSignFinish) (rsip_ctrl_t *const p_ctrl, uint8_t *const p_mac)

Finalizes a HMAC generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_mac | Pointer to destination of message digest. The length depends on MAC type. |

### ◆ hmacVerifyFinish

fsp_err_t(* rsip_api_t::hmacVerifyFinish) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_mac, uint32_t const mac_length)

Finalizes a HMAC verification.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_mac | Pointer to MAC. The length depends on mac_length. |
| [in] | mac_length | Byte length of MAC. |

### ◆ hmacSuspend

fsp_err_t(* rsip_api_t::hmacSuspend) (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t *const p_handle)

Suspend HMAC generation. This API allows you to suspend processing, for example, if you are in the middle of computing HMAC for successive chunks of the message and need to perform another process.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_handle | Pointer to destination of HMAC control block. |

### ◆ hmacResume

fsp_err_t(* rsip_api_t::hmacResume) (rsip_ctrl_t *const p_ctrl, rsip_hmac_handle_abstractor_t const *const p_handle)

Resume HMAC generation. This API allows you to resume a process that has been suspended by R_RSIP_HMAC_Suspend() API.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_handle | Pointer to HMAC control block. |

### ◆ pkiEcdsaCertVerify

fsp_err_t(* rsip_api_t::pkiEcdsaCertVerify) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_public_key, uint8_t const *const p_hash, uint8_t const *const p_signature)

Verifies a public key certificate with ECDSA.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_public_key | Pointer to wrapped key of ECC public key. |
| [in] | p_hash | Pointer to hash value. The length is as same as the key length. |
| [in] | p_signature | Pointer to signature (r, s). The length is twice as long as the key length. |

### ◆ pkiVerifiedCertInfoExport

fsp_err_t(* rsip_api_t::pkiVerifiedCertInfoExport) (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t *const p_verified_cert_info)

Exports verified certificate information stored in this driver.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_verified_cert_info | Pointer to certificate to be signed |

### ◆ pkiVerifiedCertInfoImport

fsp_err_t(* rsip_api_t::pkiVerifiedCertInfoImport) (rsip_ctrl_t *const p_ctrl, rsip_verified_cert_info_abstractor_t const *const p_verified_cert_info)

Imports verified certificate information.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_verified_cert_info | Pointer to certificate to be signed |

### ◆ pkiCertKeyImport

fsp_err_t(* rsip_api_t::pkiCertKeyImport) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_cert, uint32_t const cert_length, uint8_t const *const p_key_param1, uint32_t const key_param1_length, uint8_t const *const p_key_param2, uint32_t const key_param2_length, rsip_hash_type_t const hash_function, rsip_wrapped_key_t *const p_wrapped_public_key)

Wraps the public key in the verified public key certificate.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_cert | Pointer to certificate. |
| [in] | cert_length | Certificate length. |
| [in] | p_key_param1 | Pointer to start address of the public key parameter in certificate.<br>• [ECC] Qx |
| [in] | key_param1_length | Length of key_param1 stored in the certificate. |
| [in] | p_key_param2 | Pointer to start address of the public key parameter in certificate.<br>• [ECC] Qy |
| [in] | key_param2_length | Length of key_param2 stored in the certificate. |
| [in] | hash_function | The hash function used when verifying certificate signature. |
| [in,out] | p_wrapped_public_key | Pointer to wrapped key of public key in the certificate. |

### ◆ kdfShaInit

fsp_err_t(* rsip_api_t::kdfShaInit) (rsip_ctrl_t *const p_ctrl, rsip_hash_type_t const hash_type)

Prepares a SHA generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | hash_type | Generating hash type. |

### ◆ kdfShaEcdhSecretUpdate

fsp_err_t(* rsip_api_t::kdfShaEcdhSecretUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)

Inputs wrapped ECDH secret as a message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_secret | Pointer to wrapped secret. |

### ◆ kdfShaUpdate

fsp_err_t(* rsip_api_t::kdfShaUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |

### ◆ kdfShaFinish

fsp_err_t(* rsip_api_t::kdfShaFinish) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)

Finalizes a SHA operation and generate DKM (Derived Keying Material).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_wrapped_dkm | Pointer to destination of wrapped DKM. |

### ◆ kdfShaSuspend

fsp_err_t(* rsip_api_t::kdfShaSuspend) (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t *const p_handle)

Suspend SHA generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_handle | Pointer to destination of KDF SHA control block. |

### ◆ kdfShaResume

fsp_err_t(* rsip_api_t::kdfShaResume) (rsip_ctrl_t *const p_ctrl, rsip_kdf_sha_handle_abstractor_t const *const p_handle)

Resume SHA generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_handle | Pointer to KDF SHA control block. |

### ◆ kdfHmacDkmKeyImport

fsp_err_t(* rsip_api_t::kdfHmacDkmKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const key_length, rsip_wrapped_key_t *const p_wrapped_key)

Converts wrapped Derived Keying Material (DKM) to HMAC key for KDF.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_dkm | Pointer to wrapped DKM. |
| [in] | key_length | Length of HMAC key to be extracted from DKM. |
| [in,out] | p_wrapped_key | Pointer to destination wrapped HMAC key for KDF. |

### ◆ kdfHmacEcdhSecretKeyImport

fsp_err_t(* rsip_api_t::kdfHmacEcdhSecretKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret, rsip_wrapped_key_t *const p_wrapped_key)

Converts wrapped ECDH secret to wrapped HMAC key for KDF.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_secret | Pointer to wrapped secret. |
| [in,out] | p_wrapped_key | Pointer to destination wrapped HMAC key for KDF. |

### ◆ kdfHmacInit

fsp_err_t(* rsip_api_t::kdfHmacInit) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_key_t const *const p_wrapped_key)

Prepares a HMAC generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_key | Pointer to wrapped key of HMAC key. |

### ◆ kdfHmacDkmUpdate

fsp_err_t(* rsip_api_t::kdfHmacDkmUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm)

Inputs wrapped Derived Keying Material (DKM) as a message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_dkm | Pointer to wrapped DKM. |

### ◆ kdfHmacEcdhSecretUpdate

fsp_err_t(* rsip_api_t::kdfHmacEcdhSecretUpdate) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_secret_abstractor_t const *const p_wrapped_secret)

Inputs wrapped ECDH secret as a message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_secret | Pointer to wrapped secret. |

### ◆ kdfHmacUpdate

fsp_err_t(* rsip_api_t::kdfHmacUpdate) (rsip_ctrl_t *const p_ctrl, uint8_t const *const p_message, uint32_t const message_length)

Inputs message.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_message | Pointer to message. The length is message_length. |
| [in] | message_length | Byte length of message (0 or more bytes). |

### ◆ kdfHmacSignFinish

fsp_err_t(* rsip_api_t::kdfHmacSignFinish) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t *const p_wrapped_dkm)

Finalizes a HMAC generation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_wrapped_dkm | Pointer to destination of wrapped Derived Keying Material (DKM). |

### ◆ kdfHmacSuspend

fsp_err_t(* rsip_api_t::kdfHmacSuspend) (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t *const p_handle)

Suspends HMAC operation.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [out] | p_handle | Pointer to destination of KDF HMAC control block. |

### ◆ kdfHmacResume

fsp_err_t(* rsip_api_t::kdfHmacResume) (rsip_ctrl_t *const p_ctrl, rsip_kdf_hmac_handle_abstractor_t const *const p_handle)

Resumes HMAC operation suspended by R_RSIP_KDF_HMAC_Suspend().

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_handle | Pointer to KDF HMAC control block. |

### ◆ kdfDkmConcatenate

fsp_err_t(* rsip_api_t::kdfDkmConcatenate) (rsip_wrapped_dkm_t *const p_wrapped_dkm1, rsip_wrapped_dkm_t const *const p_wrapped_dkm2, uint32_t const wrapped_dkm1_buffer_length)

Concatenates two wrapped derived keying materials (DKMs).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_wrapped_dkm1 | Pointer to first DKM (DKM1). |
| [in] | p_wrapped_dkm2 | Pointer to second DKM (DKM2). |
| [in] | wrapped_dkm1_buffer_length | Length of wrapped_dkm1 buffer. It must be equal to or greater than DKM1 \|\| DKM2. |

### ◆ kdfDerivedKeyImport

fsp_err_t(* rsip_api_t::kdfDerivedKeyImport) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, uint32_t const position, rsip_wrapped_key_t *const p_wrapped_key)

Outputs a wrapped key from Derived Keying Material (DKM).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_dkm | Pointer to wrapped DKM. |
| [in] | position | Start position of output key value in the DKM. |
| [in,out] | p_wrapped_key | Pointer to destination of wrapped key. |

### ◆ kdfDerivedIvWrap

fsp_err_t(* rsip_api_t::kdfDerivedIvWrap) (rsip_ctrl_t *const p_ctrl, rsip_wrapped_dkm_t const *const p_wrapped_dkm, rsip_initial_vector_type_t const initial_vector_type, uint32_t const position, uint8_t const *const p_tls_sequence_num, uint8_t *const p_wrapped_initial_vector)

Outputs a initial vector from Derived Keying Material (DKM).

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | p_wrapped_dkm | Pointer to wrapped DKM. |
| [in] | initial_vector_type | Initial vector type. |
| [in] | position | Start position of output data value in the DKM. |
| [in] | p_tls_sequence_num | TLS sequence number. This argument is valid only for TLS-related data type. |
| [out] | p_wrapped_initial_vector | Pointer to destination of wrapped initial vector. |

### ◆ otfInit

| fsp_err_t(* rsip_api_t::otfInit) (rsip_ctrl_t *const p_ctrl, rsip_otf_channel_t const channel, rsip_wrapped_key_t *const p_wrapped_key, uint8_t const *const p_seed) |
|---|

Initialize on-the-fly decryption on RSIP.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. |
| [in] | channel | Channel number. |
| [in] | p_wrapped_key | Pointer to wrapped AES key. |
| [in] | p_seed | Pointer to seed. |

### ◆ rsip_instance_t

| struct rsip_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rsip_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rsip_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rsip_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rsip_sha_handle_abstractor_t

| typedef void rsip_sha_handle_abstractor_t |
|---|
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_hmac_handle_abstractor_t

| typedef void rsip_hmac_handle_abstractor_t |
|---|
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_kdf_sha_handle_abstractor_t

| typedef void rsip_kdf_sha_handle_abstractor_t |
|---|
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_kdf_hmac_handle_abstractor_t

| typedef void rsip_kdf_hmac_handle_abstractor_t |
| --- |
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_wrapped_secret_abstractor_t

| typedef void rsip_wrapped_secret_abstractor_t |
| --- |
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_verified_cert_info_abstractor_t

| typedef void rsip_verified_cert_info_abstractor_t |
| --- |
| Allocate an instance specific handle to pass into the API calls. |

### ◆ rsip_ctrl_t

| typedef void rsip_ctrl_t |
| --- |
| RSIP Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

### ◆ rsip_key_type_t

| enum rsip_key_type_t | |
|---|---|
| Key types | |
| Enumerator | |
| RSIP_KEY_TYPE_INVALID | Invalid key. |
| RSIP_KEY_TYPE_AES_128 | AES-128. |
| RSIP_KEY_TYPE_AES_192 | AES-192. |
| RSIP_KEY_TYPE_AES_256 | AES-256. |
| RSIP_KEY_TYPE_XTS_AES_128 | XTS-AES-128. |
| RSIP_KEY_TYPE_XTS_AES_256 | XTS-AES-256. |
| RSIP_KEY_TYPE_CHACHA20 | ChaCha20. |
| RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC | secp256r1 public key (also known as NIST P-256, prime256v1) |
| RSIP_KEY_TYPE_ECC_SECP384R1_PUBLIC | secp384r1 public key (also known as NIST P-384) |
| RSIP_KEY_TYPE_ECC_SECP521R1_PUBLIC | secp521r1 public key (also known as NIST P-521) |
| RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC | secp256k1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PUBLIC | brainpoolP256r1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PUBLIC | brainpoolP384r1 public key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PUBLIC | brainpoolP512r1 public key |
| RSIP_KEY_TYPE_ECC_EDWARDS25519_PUBLIC | edwards25519 public key |
| RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE | secp256r1 private key (also known as NIST P-256, prime256v1) |
| RSIP_KEY_TYPE_ECC_SECP384R1_PRIVATE | secp384r1 private key (also known as NIST P-384) |
| RSIP_KEY_TYPE_ECC_SECP521R1_PRIVATE | secp521r1 private key (also known as NIST P-521) |
| RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE | |

| | |
|---|---|
| | secp256k1 private key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP256R1_PRIVATE | brainpoolP256r1 private key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP384R1_PRIVATE | brainpoolP384r1 private key |
| RSIP_KEY_TYPE_ECC_BRAINPOOLP512R1_PRIVATE | brainpoolP512r1 private key |
| RSIP_KEY_TYPE_ECC_EDWARDS25519_PRIVATE | edwards25519 private key |
| RSIP_KEY_TYPE_RSA_1024_PUBLIC | RSA-1024 public key. |
| RSIP_KEY_TYPE_RSA_2048_PUBLIC | RSA-2048 public key. |
| RSIP_KEY_TYPE_RSA_3072_PUBLIC | RSA-3072 public key. |
| RSIP_KEY_TYPE_RSA_4096_PUBLIC | RSA-4096 public key. |
| RSIP_KEY_TYPE_RSA_1024_PRIVATE | RSA-1024 private key. |
| RSIP_KEY_TYPE_RSA_2048_PRIVATE | RSA-2048 private key. |
| RSIP_KEY_TYPE_RSA_3072_PRIVATE | RSA-3072 private key. |
| RSIP_KEY_TYPE_RSA_4096_PRIVATE | RSA-4096 private key. |
| RSIP_KEY_TYPE_HMAC_SHA1 | HMAC-SHA1. |
| RSIP_KEY_TYPE_HMAC_SHA224 | HMAC-SHA224. |
| RSIP_KEY_TYPE_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_KEY_TYPE_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_KEY_TYPE_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_KEY_TYPE_HMAC_SHA512_224 | HMAC-SHA512/224. |
| RSIP_KEY_TYPE_HMAC_SHA512_256 | HMAC-SHA512/256. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA256 | HMAC-SHA256. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA384 | HMAC-SHA384. |
| RSIP_KEY_TYPE_KDF_HMAC_SHA512 | HMAC-SHA512. |
| RSIP_KEY_TYPE_KUK | Key Update Key (KUK) |

### ◆ rsip_aes_cipher_mode_t

| enum rsip_aes_cipher_mode_t | |
|---|---|
| Block cipher modes of operation for AES | |
| Enumerator | |
| RSIP_AES_CIPHER_MODE_ECB_ENC | Electronic Codebook (ECB) mode encryption. |
| RSIP_AES_CIPHER_MODE_ECB_DEC | Electronic Codebook (ECB) mode decryption. |
| RSIP_AES_CIPHER_MODE_CBC_ENC | Cipher Block Chaining (CBC) mode encryption. |
| RSIP_AES_CIPHER_MODE_CBC_DEC | Cipher Block Chaining (CBC) mode decryption. |
| RSIP_AES_CIPHER_MODE_CTR | Counter (CTR) mode encryption or decryption. |
| RSIP_AES_CIPHER_MODE_XTS_ENC | XEX-based tweaked-codebook mode with ciphertext stealing (XTS) encryption. |
| RSIP_AES_CIPHER_MODE_XTS_DEC | XEX-based tweaked-codebook mode with ciphertext stealing (XTS) decryption. |
| RSIP_AES_CIPHER_MODE_CBC_ENC_WRAPPED_IV | Cipher Block Chaining (CBC) mode encryption with wrapped IV. |
| RSIP_AES_CIPHER_MODE_CBC_DEC_WRAPPED_IV | Cipher Block Chaining (CBC) mode decryption with wrapped IV. |

### ◆ rsip_aes_aead_mode_t

| enum rsip_aes_aead_mode_t | |
|---|---|
| AEAD modes of operation for AES | |
| Enumerator | |
| RSIP_AES_AEAD_MODE_GCM_ENC | Galois/Counter Mode (GCM) encryption. |
| RSIP_AES_AEAD_MODE_GCM_DEC | Galois/Counter Mode (GCM) decryption. |
| RSIP_AES_AEAD_MODE_CCM_ENC | Counter with CBC-MAC (CCM) encryption. |
| RSIP_AES_AEAD_MODE_CCM_DEC | Counter with CBC-MAC (CCM) decryption. |
| RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV | Galois/Counter Mode (GCM) encryption with wrapped IV. |
| RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV | Galois/Counter Mode (GCM) decryption with wrapped IV. |

### ◆ rsip_chacha_poly_mode_t

| enum rsip_chacha_poly_mode_t | |
|---|---|
| ChaCha20-Poly1305 mode | |
| Enumerator | |
| RSIP_CHACHA20_POLY1305_MODE_ENC | ChaCha20-Poly1305 encryption. |
| RSIP_CHACHA20_POLY1305_MODE_DEC | ChaCha20-Poly1305 decryption. |

### ◆ rsip_aes_mac_mode_t

| enum rsip_aes_mac_mode_t | |
|---|---|
| MAC modes of operation for AES | |
| Enumerator | |
| RSIP_AES_MAC_MODE_CMAC | Cipher-based MAC (CMAC) |

### ◆ rsip_hash_type_t

| enum rsip_hash_type_t | |
|---|---|
| Hash type | |
| Enumerator | |
| RSIP_HASH_TYPE_SHA1 | SHA-1. |
| RSIP_HASH_TYPE_SHA224 | SHA-224. |
| RSIP_HASH_TYPE_SHA256 | SHA-256. |
| RSIP_HASH_TYPE_SHA384 | SHA-384. |
| RSIP_HASH_TYPE_SHA512 | SHA-512. |
| RSIP_HASH_TYPE_SHA512_224 | SHA-512/224. |
| RSIP_HASH_TYPE_SHA512_256 | SHA-512/256. |
| RSIP_HASH_TYPE_SHA3_224 | SHA3-224. |
| RSIP_HASH_TYPE_SHA3_256 | SHA3-256. |
| RSIP_HASH_TYPE_SHA3_384 | SHA3-384. |
| RSIP_HASH_TYPE_SHA3_512 | SHA3-512. |

### ◆ rsip_mgf_type_t

| enum rsip_mgf_type_t | |
|---|---|
| MGF type | |
| Enumerator | |
| RSIP_MGF_TYPE_MGF1_SHA1 | MGF1 with SHA-1. |
| RSIP_MGF_TYPE_MGF1_SHA224 | MGF1 with SHA-224. |
| RSIP_MGF_TYPE_MGF1_SHA256 | MGF1 with SHA-256. |
| RSIP_MGF_TYPE_MGF1_SHA384 | MGF1 with SHA-384. |
| RSIP_MGF_TYPE_MGF1_SHA512 | MGF1 with SHA-512. |
| RSIP_MGF_TYPE_MGF1_SHA512_224 | MGF1 with SHA-512/224. |
| RSIP_MGF_TYPE_MGF1_SHA512_256 | MGF1 with SHA-512/256. |

### ◆ rsip_rsa_salt_length_t

| enum rsip_rsa_salt_length_t | |
|---|---|
| RSA salt length | |
| Enumerator | |
| RSIP_RSA_SALT_LENGTH_AUTO | When signing, the salt length is set to RSIP_RSA_SALT_LENGTH_MAX or RSIP_RSA_SALT_LENGTH_HASH, whichever is shorter. When verifying, the salt length is detected automatically. |
| RSIP_RSA_SALT_LENGTH_HASH | The salt length is set to the hash length. |
| RSIP_RSA_SALT_LENGTH_MAX | The salt length is set to emLen - hLen - 2, where emLen is the same as the key length and hLen is the hash length. |

### ◆ rsip_initial_vector_type_t

| enum rsip_initial_vector_type_t | |
|---|---|
| IV data types | |
| Enumerator | |
| RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE | Wrapped AES initial vector (unprocessed) |
| RSIP_INITIAL_VECTOR_TYPE_AES_TLS12_AEAD | Wrapped AES nonce (TLS1.2 AES AEAD) |
| RSIP_INITIAL_VECTOR_TYPE_AES_TLS13_AEAD | Wrapped AES nonce (TLS1.3 AES AEAD) |

### ◆ rsip_otf_channel_t

| enum rsip_otf_channel_t | |
|---|---|
| Channel number of on-the-fly decryption/encryption module. | |
| Enumerator | |
| RSIP_OTF_CHANNEL_0 | Channel 0. |
| RSIP_OTF_CHANNEL_1 | Channel 1. |
| RSIP_OTF_CHANNEL_2 | Channel 2. |

## 5.3.13.2 RSIP key injection Interface

Interfaces » Security

### Detailed Description

Interface for key injection by Renesas Secure IP (RSIP) functions.

# Summary

The RSIP key injection interface provides RSIP functionality.

### Data Structures

| | |
|---|---|
| struct | rsip_aes_wrapped_key_t |
| struct | rsip_chacha20_wrapped_key_t |

| | |
|---|---|
| struct | rsip_rsa2048_private_wrapped_key_t |
| struct | rsip_rsa3072_private_wrapped_key_t |
| struct | rsip_rsa4096_private_wrapped_key_t |
| struct | rsip_ecc_private_wrapped_key_t |
| struct | rsip_hmac_sha_wrapped_key_t |
| struct | rsip_key_injection_api_t |

### Enumerations

| | |
|---|---|
| enum | rsip_key_injection_type_t |

## Data Structure Documentation

### ◆ rsip_aes_wrapped_key_t

| struct rsip_aes_wrapped_key_t | | |
|---|---|---|
| AES128/256 wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_AES256_XTS_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_chacha20_wrapped_key_t

| struct rsip_chacha20_wrapped_key_t | | |
|---|---|---|
| CHACHA20 wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_CHACHA20_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_rsa2048_private_wrapped_key_t

| struct rsip_rsa2048_private_wrapped_key_t | | |
|---|---|---|
| RSA 2048bit private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_RSA2048_PRIVATE_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_rsa3072_private_wrapped_key_t

| struct rsip_rsa3072_private_wrapped_key_t | | |
|---|---|---|
| RSA 3072bit private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_RSA3072_PRIVATE_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_rsa4096_private_wrapped_key_t

| struct rsip_rsa4096_private_wrapped_key_t | | |
|---|---|---|
| RSA 4096bit private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_RSA4096_PRIVATE_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_ecc_private_wrapped_key_t

| struct rsip_ecc_private_wrapped_key_t | | |
|---|---|---|
| ECC 256/384/512/521 private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_ECCP521_PRIVATE_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_hmac_sha_wrapped_key_t

| struct rsip_hmac_sha_wrapped_key_t | | |
|---|---|---|
| HMAC 256/342/512 wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| uint32_t | value[R_RSIP_HMAC_SHA512_KEY_INDEX_WORD_SIZE] | Wrapped key value. |

### ◆ rsip_key_injection_api_t

| struct rsip_key_injection_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | AES128_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |

| | |
|---|---|
| | |
| fsp_err_t(* | AES192_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | AES256_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | AES128_XTS_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | AES256_XTS_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ChaCha20_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | RSA3072_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | RSA4096_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const |

| | |
|---|---|
| | p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp521r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256k1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP256r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP384r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP512r1_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | ECC_Ed25519_InitialPrivateKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA224_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA256_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA384_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA512_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA512_224_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |
| fsp_err_t(* | HMAC_SHA512_256_InitialKeyWrap )(rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key) |
| | |

## Field Documentation

### ◆ AES128_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES128_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 128-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 128-bit AES wrapped key |

### ◆ AES192_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES192_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 192-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 192-bit AES wrapped key |

### ◆ AES256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit AES wrapped key |

### ◆ AES128_XTS_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES128_XTS_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 128-bit AES-XTS wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 128-bit AES-XTS wrapped key |

### ◆ AES256_XTS_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::AES256_XTS_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_aes_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit AES-XTS wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit AES-XTS wrapped key |

### ◆ ChaCha20_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ChaCha20_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_chacha20_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ChaCha20 wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit ChaCha20 wrapped key |

### ◆ RSA2048_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA2048_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa2048_private_wrapped_key_t *const p_wrapped_key)

This API outputs 2048-bit RSA wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 2048-bit RSA wrapped private key |

### ◆ RSA3072_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA3072_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa3072_private_wrapped_key_t *const p_wrapped_key)

This API outputs 3072-bit RSA wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 3072-bit RSA wrapped private key |

### ◆ RSA4096_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::RSA4096_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_rsa4096_private_wrapped_key_t *const p_wrapped_key)

This API outputs 4096-bit RSA wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 4096-bit RSA wrapped private key |

### ◆ ECC_secp256r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp256r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

### ◆ ECC_secp384r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp384r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 384-bit ECC wrapped private key |

### ◆ ECC_secp521r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp521r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 521-bit ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 521-bit ECC wrapped private key |

### ◆ ECC_secp256k1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_secp256k1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

### ◆ ECC_brainpoolP256r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP256r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit brainpool ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

### ◆ ECC_brainpoolP384r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP384r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit brainpool ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 384-bit ECC wrapped private key |

### ◆ ECC_brainpoolP512r1_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_brainpoolP512r1_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 512-bit brainpool ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 512-bit ECC wrapped private key |

### ◆ ECC_Ed25519_InitialPrivateKeyWrap

fsp_err_t(* rsip_key_injection_api_t::ECC_Ed25519_InitialPrivateKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_ecc_private_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit Edwards curve ECC wrapped private key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit ECC wrapped private key |

### ◆ HMAC_SHA224_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA224_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 224-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 224-bit HMAC-SHA wrapped key |

### ◆ HMAC_SHA256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 256-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 256-bit HMAC-SHA wrapped key |

### ◆ HMAC_SHA384_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA384_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 384-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 384-bit HMAC-SHA wrapped key |

### ◆ HMAC_SHA512_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 512-bit HMAC-SHA wrapped key |

### ◆ HMAC_SHA512_224_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_224_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512/224-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 512/224-bit HMAC-SHA wrapped key |

### ◆ HMAC_SHA512_256_InitialKeyWrap

fsp_err_t(* rsip_key_injection_api_t::HMAC_SHA512_256_InitialKeyWrap) (rsip_key_injection_type_t const key_injection_type, uint8_t const *const p_wrapped_user_factory_programming_key, uint8_t const *const p_initial_vector, uint8_t const *const p_user_key, rsip_hmac_sha_wrapped_key_t *const p_wrapped_key)

This API outputs 512/256-bit HMAC-SHA wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_injection_type | Selection key injection type when generating wrapped key |
| [in] | p_wrapped_user_factory_programming_key | Wrapped user factory programming key by the Renesas Key Wrap Service. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_initial_vector | Initialization vector when generating encrypted key. When key injection type is plain, this is not required and any value can be specified. |
| [in] | p_user_key | User key. If key injection type is not plain, it must be encrypted and have MAC appended. |
| [out] | p_wrapped_key | 512/256-bit HMAC-SHA wrapped key |

## Enumeration Type Documentation

### ◆ rsip_key_injection_type_t

enum rsip_key_injection_type_t

Key injection type.

| Enumerator | |
|---|---|
| RSIP_KEY_INJECTION_TYPE_ENCRYPTED | Input encrypted user key. |
| RSIP_KEY_INJECTION_TYPE_PLAIN | Input plain user key. |

### 5.3.13.3 SCE Interface

Interfaces » Security

#### Detailed Description

Interface for Secure Crypto Engine (SCE) functions.

# Summary

The SCE interface provides SCE functionality.

The SCE interface can be implemented by:

- SCE Protected Mode

#### Data Structures

| | |
|---|---|
| struct | sce_byte_data_t |
| struct | sce_aes_wrapped_key_t |
| struct | sce_hmac_sha_wrapped_key_t |
| struct | sce_rsa1024_public_wrapped_key_t |
| struct | sce_rsa1024_private_wrapped_key_t |
| struct | sce_rsa2048_public_wrapped_key_t |
| struct | sce_rsa2048_private_wrapped_key_t |
| struct | sce_rsa3072_public_wrapped_key_t |
| struct | sce_rsa4096_public_wrapped_key_t |
| struct | sce_rsa1024_wrapped_pair_key_t |
| struct | sce_rsa2048_wrapped_pair_key_t |
| struct | sce_ecc_public_wrapped_key_t |
| struct | sce_ecc_private_wrapped_key_t |
| struct | sce_ecc_wrapped_pair_key_t |
| struct | sce_ecdh_wrapped_key_t |
| struct | sce_key_update_key_t |

| | |
|---|---|
| struct | sce_aes_handle_t |
| struct | sce_gcm_handle_t |
| struct | sce_ccm_handle_t |
| struct | sce_cmac_handle_t |
| struct | sce_sha_md5_handle_t |
| struct | sce_hmac_sha_handle_t |
| struct | sce_ecdh_handle_t |
| struct | sce_cfg_t |
| struct | sce_api_t |
| struct | sce_instance_t |

### Typedefs

| | |
|---|---|
| typedef sce_byte_data_t | sce_rsa_byte_data_t<br>byte data More... |
| typedef sce_byte_data_t | sce_ecdsa_byte_data_t<br>byte data More... |
| typedef void | sce_ctrl_t |

## Data Structure Documentation

### ◆ sce_byte_data_t

| struct sce_byte_data_t | | |
|---|---|---|
| Byte data structure | | |
| Data Fields | | |
| uint8_t * | pdata | pointer |
| uint32_t | data_length | data_length |
| uint32_t | data_type | data type |

### ◆ sce_aes_wrapped_key_t

| struct sce_aes_wrapped_key_t |
|---|

| AES wrapped key data structure. DO NOT MODIFY. | | |
|---|---|---|
| Data Fields | | |
| uint32_t | type | key type |
| uint32_t | value[SCE_TLS_AES256_KEY_INDEX_WORD_SIZE] | wrapped key value<br><br>*Note*<br>*The size of the "value" array and the definition name depends on the used SCE. See the header file for detail.* |

### ◆ sce_hmac_sha_wrapped_key_t

| struct sce_hmac_sha_wrapped_key_t | | |
|---|---|---|
| HMAC-SHA wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | key type |
| uint32_t | value[SCE_TLS_HMAC_KEY_INDEX_WORD_SIZE] | wrapped key value |

### ◆ sce_rsa1024_public_wrapped_key_t

| struct sce_rsa1024_public_wrapped_key_t | | |
|---|---|---|
| RSA 1024bit public wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | key type |
| struct sce_rsa1024_public_wrapped_key_t | value | |

### ◆ sce_rsa1024_private_wrapped_key_t

| struct sce_rsa1024_private_wrapped_key_t | | |
|---|---|---|
| RSA 1024bit private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | key type |
| struct sce_rsa1024_private_wrapped_key_t | value | |

### ◆ sce_rsa2048_public_wrapped_key_t

| struct sce_rsa2048_public_wrapped_key_t |
|---|
| RSA 2048bit public wrapped key data structure. DO NOT MODIFY. |

| Data Fields | | |
|---|---|---|
| uint32_t | type | Key type. |
| struct sce_rsa2048_public_wrapped_key_t | value | |

### ◆ sce_rsa2048_private_wrapped_key_t

| struct sce_rsa2048_private_wrapped_key_t | | |
|---|---|---|
| RSA 2048bit private wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | key type |
| struct sce_rsa2048_private_wrapped_key_t | value | |

### ◆ sce_rsa3072_public_wrapped_key_t

| struct sce_rsa3072_public_wrapped_key_t | | |
|---|---|---|
| RSA 3072bit public wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| struct sce_rsa3072_public_wrapped_key_t | value | |

### ◆ sce_rsa4096_public_wrapped_key_t

| struct sce_rsa4096_public_wrapped_key_t | | |
|---|---|---|
| RSA 4096bit public wrapped key data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | Key type. |
| struct sce_rsa4096_public_wrapped_key_t | value | |

### ◆ sce_rsa1024_wrapped_pair_key_t

| struct sce_rsa1024_wrapped_pair_key_t | | |
|---|---|---|
| RSA 1024bit wrapped key pair structure. DO NOT MODIFY. | | |
| Data Fields | | |
| sce_rsa1024_private_wrapped_key_t | priv_key | RSA 1024-bit private wrapped key. |
| sce_rsa1024_public_wrapped_k | pub_key | RSA 1024-bit public wrapped |

| | | |
|---|---|---|
| ey_t | | key. |

◆ **sce_rsa2048_wrapped_pair_key_t**

| struct sce_rsa2048_wrapped_pair_key_t | | |
|---|---|---|
| RSA 2048bit wrapped key pair structure. DO NOT MODIFY. | | |
| Data Fields | | |
| sce_rsa2048_private_wrapped_key_t | priv_key | RSA 2048-bit private wrapped key. |
| sce_rsa2048_public_wrapped_key_t | pub_key | RSA 2048-bit public wrapped key. |

◆ **sce_ecc_public_wrapped_key_t**

| struct sce_ecc_public_wrapped_key_t | | |
|---|---|---|
| ECC P-192/224/256 public wrapped key data structure | | |
| Data Fields | | |
| uint32_t | type | key type |
| struct sce_ecc_public_wrapped_key_t | value | |

◆ **sce_ecc_private_wrapped_key_t**

| struct sce_ecc_private_wrapped_key_t | | |
|---|---|---|
| ECC P-192/224/256 private wrapped key data structure | | |
| Data Fields | | |
| uint32_t | type | key type |
| uint32_t | value[HW_SCE_ECC_PRIVATE_KEY_MANAGEMENT_INFO_WORD_SIZE] | wrapped key value |

◆ **sce_ecc_wrapped_pair_key_t**

| struct sce_ecc_wrapped_pair_key_t | | |
|---|---|---|
| ECC P-192/224/256 wrapped key pair structure | | |
| Data Fields | | |
| sce_ecc_private_wrapped_key_t | priv_key | ECC private wrapped key. |
| sce_ecc_public_wrapped_key_t | pub_key | ECC public wrapped key. |

◆ **sce_ecdh_wrapped_key_t**

| struct sce_ecdh_wrapped_key_t | | |
|---|---|---|
| ECDH wrapped key data structure | | |
| Data Fields | | |
| | | |

| | | |
|---|---|---|
| uint32_t | type | key type |
| uint32_t | value[HW_SCE_SHARED_SECRET_KEY_INDEX_WORD_SIZE] | wrapped key value |

### ◆ sce_key_update_key_t

| struct sce_key_update_key_t | | |
|---|---|---|
| Update key ring index data structure. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | type | key type |
| uint32_t | value[HW_SCE_UPDATE_KEY_RING_INDEX_WORD_SIZE] | wrapped key value<br><br>*Note*<br>*The size of the "value" array and the definition name depends on the used SCE. See the header file for detail.* |

### ◆ sce_aes_handle_t

| struct sce_aes_handle_t | | |
|---|---|---|
| The work area for AES. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| sce_aes_wrapped_key_t | wrapped_key | wrapped key |
| uint32_t | current_input_data_size | text size under encryption / decryption |
| uint8_t | last_1_block_as_fraction[HW_SCE_AES_BLOCK_BYTE_SIZE] | text array less than the block long |
| uint8_t | last_2_block_as_fraction[HW_SCE_AES_BLOCK_BYTE_SIZE *2] | reserved |
| uint8_t | current_initial_vector[HW_SCE_AES_CBC_IV_BYTE_SIZE] | current initialization vector used in CBC mode |
| uint8_t | flag_call_init | control flag of calling function |

### ◆ sce_gcm_handle_t

| struct sce_gcm_handle_t | | |
|---|---|---|
| The work area for GCM. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| sce_aes_wrapped_key_t | wrapped_key | wrapped key |
| uint8_t | gcm_buffer[HW_SCE_AES_BLOC | text array less than the block |

| | | |
|---|---|---|
| | K_BYTE_SIZE] | long |
| uint8_t | gcm_aad_buffer[HW_SCE_AES_GCM_AAD_BLOCK_BYTE_SIZE] | AAD array less than the block long. |
| uint32_t | all_received_length | entire length of text |
| uint32_t | all_received_aad_length | entire length of text |
| uint32_t | buffering_length | text array length less than the block long |
| uint32_t | buffering_aad_length | AAD array length less than the block long. |
| uint8_t | flag_call_init | control flag of calling function |
| uint8_t | flag_update_input_data | control flag of next input data |

### ◆ sce_ccm_handle_t

| struct sce_ccm_handle_t | | |
|---|---|---|
| The work area for CCM. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| sce_aes_wrapped_key_t | wrapped_key | wrapped key |
| uint8_t | formatted_data[HW_SCE_AES_CCM_B_FORMAT_BYTE_SIZE] | formatted data area |
| uint8_t | counter[HW_SCE_AES_CCM_COUNTER_BYTE_SIZE] | counter of CTR mode |
| uint8_t | ccm_buffer[HW_SCE_AES_BLOCK_BYTE_SIZE] | text array less than the block long |
| uint32_t | all_received_length | entire length of text |
| uint32_t | buffering_length | text array length less than the block long |
| uint8_t | flag_call_init | control flag of calling function |

### ◆ sce_cmac_handle_t

| struct sce_cmac_handle_t | | |
|---|---|---|
| The work area for CMAC. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| sce_aes_wrapped_key_t | wrapped_key | wrapped key |
| uint8_t | cmac_buffer[HW_SCE_AES_BLOCK_BYTE_SIZE] | message array less than the block long |
| uint32_t | all_received_length | entire length of message |
| | | |

| | | |
|---|---|---|
| uint32_t | buffering_length | message array length less than the block long |
| uint8_t | flag_call_init | control flag of calling function |

### ◆ sce_sha_md5_handle_t

| struct sce_sha_md5_handle_t | | |
|---|---|---|
| The work area for SHA. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| uint8_t | sha_buffer[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE *4] | message array length less than the block long |
| uint32_t | all_received_length | entire length of message |
| uint32_t | buffering_length | message array length less than the block long |
| uint8_t | current_hash[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE] | last hash value |
| uint8_t | flag_call_init | control flag of calling function |

### ◆ sce_hmac_sha_handle_t

| struct sce_hmac_sha_handle_t | | |
|---|---|---|
| The work area for HMAC-SHA. DO NOT MODIFY. | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| sce_hmac_sha_wrapped_key_t | wrapped_key | wrapped key |
| uint8_t | hmac_buffer[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE *4] | message array length less than the block long |
| uint32_t | all_received_length | entire length of message |
| uint32_t | buffering_length | message array length less than the block long |
| uint8_t | flag_call_init | control flag of calling function |

### ◆ sce_ecdh_handle_t

| struct sce_ecdh_handle_t | | |
|---|---|---|
| The work area for ECDH | | |
| Data Fields | | |
| uint32_t | id | serial number of this handle |
| uint32_t | flag_use_key_id | control frag that the key_id has already used or not |
| uint32_t | key_id | serial number of the wrapped |

| | | |
|---|---|---|
| | | key |
| uint32_t | key_type | key type |
| uint8_t | flag_call_init | control flag of calling function |
| uint8_t | flag_call_make_public | control flag of calling function |
| uint8_t | flag_call_read_public | control flag of calling function |
| uint8_t | flag_call_shared_secret | control flag of calling function |

### ◆ sce_cfg_t

| struct sce_cfg_t |
|---|
| User configuration structure, used in open function |

### ◆ sce_api_t

| struct sce_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(sce_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | softwareReset )(void) |
| | |
| fsp_err_t(* | randomNumberGenerate )(uint32_t *random) |
| | |
| fsp_err_t(* | AES128_WrappedKeyGenerate )(sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256_WrappedKeyGenerate )(sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES128_EncryptedKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256_EncryptedKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | AES128_RFC3394KeyWrap )(sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key) |
| | |
| fsp_err_t(* | AES256_RFC3394KeyWrap )(sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key) |
| | |
| fsp_err_t(* | AES128_RFC3394KeyUnwrap )(sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key) |
| | |
| fsp_err_t(* | AES256_RFC3394KeyUnwrap )(sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key) |
| | |
| fsp_err_t(* | AES128ECB_EncryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES128ECB_EncryptUpdate )(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |
| fsp_err_t(* | AES128ECB_EncryptFinal )(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| | |
| fsp_err_t(* | AES128ECB_DecryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES128ECB_DecryptUpdate )(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES128ECB_DecryptFinal )(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| | |
| fsp_err_t(* | AES256ECB_EncryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256ECB_EncryptUpdate )(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |

| | |
|---|---|
| fsp_err_t(* | AES256ECB_EncryptFinal )(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| | |
| fsp_err_t(* | AES256ECB_DecryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256ECB_DecryptUpdate )(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES256ECB_DecryptFinal )(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| | |
| fsp_err_t(* | AES128CBC_EncryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| | |
| fsp_err_t(* | AES128CBC_EncryptUpdate )(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |
| fsp_err_t(* | AES128CBC_EncryptFinal )(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| | |
| fsp_err_t(* | AES128CBC_DecryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| | |
| fsp_err_t(* | AES128CBC_DecryptUpdate )(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES128CBC_DecryptFinal )(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| | |
| fsp_err_t(* | AES256CBC_EncryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| | |
| fsp_err_t(* | AES256CBC_EncryptUpdate )(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |
| fsp_err_t(* | AES256CBC_EncryptFinal )(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length) |
| | |

| | |
|---|---|
| fsp_err_t(* | AES256CBC_DecryptInit )(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector) |
| | |
| fsp_err_t(* | AES256CBC_DecryptUpdate )(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES256CBC_DecryptFinal )(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length) |
| | |
| fsp_err_t(* | AES128GCM_EncryptInit )(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| | |
| fsp_err_t(* | AES128GCM_EncryptUpdate )(sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length) |
| | |
| fsp_err_t(* | AES128GCM_EncryptFinal )(sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag) |
| | |
| fsp_err_t(* | AES128GCM_DecryptInit )(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| | |
| fsp_err_t(* | AES128GCM_DecryptUpdate )(sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length) |
| | |
| fsp_err_t(* | AES128GCM_DecryptFinal )(sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length) |
| | |
| fsp_err_t(* | AES256GCM_EncryptInit )(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| | |
| fsp_err_t(* | AES256GCM_EncryptUpdate )(sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length) |
| | |
| fsp_err_t(* | AES256GCM_EncryptFinal )(sce_gcm_handle_t *handle, uint8_t |

| | |
|---|---|
| | *cipher, uint32_t *cipher_data_length, uint8_t *atag) |
| | |
| fsp_err_t(* | AES256GCM_DecryptInit )(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length) |
| | |
| fsp_err_t(* | AES256GCM_DecryptUpdate )(sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length) |
| | |
| fsp_err_t(* | AES256GCM_DecryptFinal )(sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length) |
| | |
| fsp_err_t(* | AES128CCM_EncryptInit )(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES128CCM_EncryptUpdate )(sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |
| fsp_err_t(* | AES128CCM_EncryptFinal )(sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES128CCM_DecryptInit )(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES128CCM_DecryptUpdate )(sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES128CCM_DecryptFinal )(sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES256CCM_EncryptInit )(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length) |
| | |

| | |
|---|---|
| fsp_err_t(* | AES256CCM_EncryptUpdate )(sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length) |
| | |
| fsp_err_t(* | AES256CCM_EncryptFinal )(sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES256CCM_DecryptInit )(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES256CCM_DecryptUpdate )(sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length) |
| | |
| fsp_err_t(* | AES256CCM_DecryptFinal )(sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES128CMAC_GenerateInit )(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES128CMAC_GenerateUpdate )(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | AES128CMAC_GenerateFinal )(sce_cmac_handle_t *handle, uint8_t *mac) |
| | |
| fsp_err_t(* | AES128CMAC_VerifyInit )(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES128CMAC_VerifyUpdate )(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | AES128CMAC_VerifyFinal )(sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | AES256CMAC_GenerateInit )(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256CMAC_GenerateUpdate )(sce_cmac_handle_t *handle, uint8_t |

| | |
|---|---|
| | *message, uint32_t message_length) |
| | |
| fsp_err_t(* | AES256CMAC_GenerateFinal )(sce_cmac_handle_t *handle, uint8_t *mac) |
| | |
| fsp_err_t(* | AES256CMAC_VerifyInit )(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | AES256CMAC_VerifyUpdate )(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | AES256CMAC_VerifyFinal )(sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | SHA256_Init )(sce_sha_md5_handle_t *handle) |
| | |
| fsp_err_t(* | SHA256_Update )(sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | SHA256_Final )(sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length) |
| | |
| fsp_err_t(* | RSA1024_WrappedKeyPairGenerate )(sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | RSA2048_WrappedKeyPairGenerate )(sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | RSA1024_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSA1024_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key) |

| | |
|---|---|
| | |
| fsp_err_t(* | RSA2048_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSA3072_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSA4096_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSASSA_PKCS1024_SignatureGenerate )(sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |
| fsp_err_t(* | RSASSA_PKCS2048_SignatureGenerate )(sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |
| fsp_err_t(* | RSASSA_PKCS1024_SignatureVerify )(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |
| fsp_err_t(* | RSASSA_PKCS2048_SignatureVerify )(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |
| fsp_err_t(* | RSASSA_PKCS3072_SignatureVerify )(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |
| fsp_err_t(* | RSASSA_PKCS4096_SignatureVerify )(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type) |
| | |

| | |
|---|---|
| fsp_err_t(* | RSAES_PKCS1024_Encrypt )(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSAES_PKCS2048_Encrypt )(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSAES_PKCS3072_Encrypt )(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSAES_PKCS4096_Encrypt )(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSAES_PKCS1024_Decrypt )(sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | RSAES_PKCS2048_Decrypt )(sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | SHA256HMAC_EncryptedKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | SHA256HMAC_GenerateInit )(sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | SHA256HMAC_GenerateUpdate )(sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | SHA256HMAC_GenerateFinal )(sce_hmac_sha_handle_t *handle, uint8_t *mac) |
| | |
| fsp_err_t(* | SHA256HMAC_VerifyInit )(sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | SHA256HMAC_VerifyUpdate )(sce_hmac_sha_handle_t *handle, |

| | |
|---|---|
| | uint8_t *message, uint32_t message_length) |
| | |
| fsp_err_t(* | SHA256HMAC_VerifyFinal )(sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length) |
| | |
| fsp_err_t(* | ECC_secp192r1_WrappedKeyPairGenerate )(sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | ECC_secp224r1_WrappedKeyPairGenerate )(sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_WrappedKeyPairGenerate )(sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_WrappedKeyPairGenerate )(sce_ecc_wrapped_pair_key_t *wrapped_pair_key) |
| | |
| fsp_err_t(* | ECC_secp192r1_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp224r1_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_EncryptedPublicKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp192r1_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp224r1_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | ECC_secp256r1_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_EncryptedPrivateKeyWrap )(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp192r1_SignatureGenerate )(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp224r1_SignatureGenerate )(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp256r1_SignatureGenerate )(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp384r1_SignatureGenerate )(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp192r1_SignatureVerify )(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp224r1_SignatureVerify )(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp256r1_SignatureVerify )(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDSA_secp384r1_SignatureVerify )(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDH_secp256r1_Init )(sce_ecdh_handle_t *handle, uint32_t |

| | |
|---|---|
| | key_type, uint32_t use_key_id) |
| | |
| fsp_err_t(* | ECDH_secp256r1_PublicKeySign )(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDH_secp256r1_PublicKeyVerify )(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDH_secp256r1_PublicKeyReadWithoutSignature )(sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | ECDH_secp256r1_SharedSecretCalculate )(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key) |
| | |
| fsp_err_t(* | ECDH_secp256r1_KeyDerivation )(sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | TLS_RootCertificateRSA2048PublicKeyInstall )(uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key) |
| | |
| fsp_err_t(* | TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate )(sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key) |
| | |
| fsp_err_t(* | TLS_RootCertificateVerify )(uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key) |
| | |

| | |
|---|---|
| fsp_err_t(* | TLS_CertificateVerify )(uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key) |
| | |
| fsp_err_t(* | TLS_PreMasterSecretGenerateForRSA2048 )(uint32_t *sce_pre_master_secret) |
| | |
| fsp_err_t(* | TLS_MasterSecretGenerate )(uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret) |
| | |
| fsp_err_t(* | TLS_PreMasterSecretEncryptWithRSA2048 )(uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret) |
| | |
| fsp_err_t(* | TLS_SessionKeyGenerate )(uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector) |
| | |
| fsp_err_t(* | TLS_VerifyDataGenerate )(uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data) |
| | |
| fsp_err_t(* | TLS_ServerKeyExchangeVerify )(uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key) |
| | |
| fsp_err_t(* | TLS_PreMasterSecretGenerateForECC_secp256r1 )(uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* sce_api_t::open) (sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg)

Enables use of SCE functionality.

**Implemented as**

- R_SCE_Open()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration structure. |

### ◆ close

fsp_err_t(* sce_api_t::close) (sce_ctrl_t *const p_ctrl)

Stops supply of power to the SCE.

**Implemented as**

- R_SCE_Close()

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ softwareReset

fsp_err_t(* sce_api_t::softwareReset) (void)

Software reset to SCE.

**Implemented as**

- R_SCE_SoftwareReset()

### ◆ randomNumberGenerate

fsp_err_t(* sce_api_t::randomNumberGenerate) (uint32_t *random)

Generates 4 words random number.

**Implemented as**

- R_SCE_RandomNumberGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | random | Stores 4words (16 bytes) random data. |

### ◆ AES128_WrappedKeyGenerate

fsp_err_t(* sce_api_t::AES128_WrappedKeyGenerate) (sce_aes_wrapped_key_t *wrapped_key)

This API outputs 128-bit AES wrapped key.

**Implemented as**

- R_SCE_AES128_WrappedKeyGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES256_WrappedKeyGenerate

fsp_err_t(* sce_api_t::AES256_WrappedKeyGenerate) (sce_aes_wrapped_key_t *wrapped_key)

This API outputs 256-bit AES wrapped key.

**Implemented as**

- R_SCE_AES256_WrappedKeyGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES128_EncryptedKeyWrap

fsp_err_t(* sce_api_t::AES128_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)

This API outputs 128-bit AES wrapped key.

**Implemented as**

- R_SCE_AES128_EncryptedKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES256_EncryptedKeyWrap

fsp_err_t(* sce_api_t::AES256_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)

This API outputs 256-bit AES wrapped key.

**Implemented as**

- R_SCE_AES256_EncryptedKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES128_RFC3394KeyWrap

fsp_err_t(* sce_api_t::AES128_RFC3394KeyWrap) (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key)

This API outputs 128-bit AES wrapped key.

**Implemented as**

- R_SCE_AES128_RFC3394KeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-128 key used for wrapping. |
| [in] | target_key_type | Selects key to be wrapped. |
| [in] | target_key | Key to be wrapped. |
| [out] | rfc3394_wrapped_key | Wrapped key. |

### ◆ AES256_RFC3394KeyWrap

fsp_err_t(* sce_api_t::AES256_RFC3394KeyWrap) (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, sce_aes_wrapped_key_t *target_key, uint32_t *rfc3394_wrapped_key)

This API outputs 256-bit AES wrapped key.

**Implemented as**

- R_SCE_AES256_RFC3394KeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-256 key used for wrapping. |
| [in] | target_key_type | Selects key to be wrapped. |
| [in] | target_key | Key to be wrapped. |
| [out] | rfc3394_wrapped_key | Wrapped key. |

### ◆ AES128_RFC3394KeyUnwrap

fsp_err_t(* sce_api_t::AES128_RFC3394KeyUnwrap) (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key)

This API outputs 128-bit AES unwrapped key.

**Implemented as**

- R_SCE_AES128_RFC3394KeyUnwrap()

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-128 key used for unwrapping. |
| [in] | target_key_type | Selects key to be unwrapped. |
| [in] | rfc3394_wrapped_key | Wrapped key. |
| [out] | target_key | Key to be unwrapped. |

### ◆ AES256_RFC3394KeyUnwrap

fsp_err_t(* sce_api_t::AES256_RFC3394KeyUnwrap) (sce_aes_wrapped_key_t *master_key, uint32_t target_key_type, uint32_t *rfc3394_wrapped_key, sce_aes_wrapped_key_t *target_key)

This API outputs 256-bit AES unwrapped key.

**Implemented as**

- R_SCE_AES256_RFC3394KeyUnwrap()

**Parameters**

| | | |
|---|---|---|
| [in] | master_key | AES-256 key used for unwrapping. |
| [in] | target_key_type | Selects key to be unwrapped. |
| [in] | rfc3394_wrapped_key | Wrapped key. |
| [out] | target_key | Key to be unwrapped. |

### ◆ AES128ECB_EncryptInit

fsp_err_t(* sce_api_t::AES128ECB_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128ECB encryption.

**Implemented as**

- R_SCE_AES128ECB_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES128ECB_EncryptUpdate

fsp_err_t(* sce_api_t::AES128ECB_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128ECB encryption.

**Implemented as**

- R_SCE_AES128ECB_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

### ◆ AES128ECB_EncryptFinal

fsp_err_t(* sce_api_t::AES128ECB_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES128ECB encryption.

**Implemented as**

- R_SCE_AES128ECB_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

### ◆ AES128ECB_DecryptInit

fsp_err_t(* sce_api_t::AES128ECB_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128ECB decryption.

**Implemented as**

- R_SCE_AES128ECB_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES128ECB_DecryptUpdate

fsp_err_t(* sce_api_t::AES128ECB_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128ECB decryption.

**Implemented as**

- R_SCE_AES128ECB_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

### ◆ AES128ECB_DecryptFinal

fsp_err_t(* sce_api_t::AES128ECB_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES128ECB decryption.

**Implemented as**

- R_SCE_AES128ECB_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

### ◆ AES256ECB_EncryptInit

fsp_err_t(* sce_api_t::AES256ECB_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256ECB encryption.

**Implemented as**

- R_SCE_AES256ECB_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES256ECB_EncryptUpdate

fsp_err_t(* sce_api_t::AES256ECB_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256ECB encryption.

**Implemented as**

- R_SCE_AES256ECB_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

### ◆ AES256ECB_EncryptFinal

fsp_err_t(* sce_api_t::AES256ECB_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES256ECB encryption.

**Implemented as**

- R_SCE_AES256ECB_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

### ◆ AES256ECB_DecryptInit

fsp_err_t(* sce_api_t::AES256ECB_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256ECB decryption.

**Implemented as**

- R_SCE_AES256ECB_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES256ECB_DecryptUpdate

fsp_err_t(* sce_api_t::AES256ECB_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256ECB decryption.

**Implemented as**

- R_SCE_AES256ECB_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

### ◆ AES256ECB_DecryptFinal

fsp_err_t(* sce_api_t::AES256ECB_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES256ECB decryption.

**Implemented as**

- R_SCE_AES256ECB_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

### ◆ AES128CBC_EncryptInit

fsp_err_t(* sce_api_t::AES128CBC_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES128CBC encryption.

**Implemented as**

- R_SCE_AES128CBC_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initial vector area (16byte) |

### ◆ AES128CBC_EncryptUpdate

fsp_err_t(* sce_api_t::AES128CBC_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128CBC encryption.

**Implemented as**

- R_SCE_AES128CBC_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

### ◆ AES128CBC_EncryptFinal

fsp_err_t(* sce_api_t::AES128CBC_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES128CBC encryption.

**Implemented as**

- R_SCE_AES128CBC_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

### ◆ AES128CBC_DecryptInit

fsp_err_t(* sce_api_t::AES128CBC_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES128CBC decryption.

**Implemented as**

- R_SCE_AES128CBC_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initial vector area (16byte) |

### ◆ AES128CBC_DecryptUpdate

fsp_err_t(* sce_api_t::AES128CBC_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128CBC decryption.

**Implemented as**

- R_SCE_AES128CBC_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

### ◆ AES128CBC_DecryptFinal

fsp_err_t(* sce_api_t::AES128CBC_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES128CBC decryption.

**Implemented as**

- R_SCE_AES128CBC_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

### ◆ AES256CBC_EncryptInit

fsp_err_t(* sce_api_t::AES256CBC_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES256CBC encryption.

**Implemented as**

- R_SCE_AES256CBC_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initial vector area (16byte) |

### ◆ AES256CBC_EncryptUpdate

fsp_err_t(* sce_api_t::AES256CBC_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256CBC encryption.

**Implemented as**

- R_SCE_AES256CBC_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in,out] | plain_length | plaintext data length (must be a multiple of 16) |

### ◆ AES256CBC_EncryptFinal

fsp_err_t(* sce_api_t::AES256CBC_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES256CBC encryption.

**Implemented as**

- R_SCE_AES256CBC_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | cipher | ciphertext data area (nothing ever written here) |
| [in,out] | cipher_length | ciphertext data length (0 always written here) |

### ◆ AES256CBC_DecryptInit

fsp_err_t(* sce_api_t::AES256CBC_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES256CBC decryption.

**Implemented as**

- R_SCE_AES256CBC_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initial vector area (16byte) |

### ◆ AES256CBC_DecryptUpdate

fsp_err_t(* sce_api_t::AES256CBC_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256CBC decryption.

**Implemented as**

- R_SCE_AES256CBC_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in,out] | cipher_length | ciphertext data length (must be a multiple of 16) |

### ◆ AES256CBC_DecryptFinal

fsp_err_t(* sce_api_t::AES256CBC_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES256CBC decryption.

**Implemented as**

- R_SCE_AES256CBC_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES handler (work area) |
| [in,out] | plain | plaintext data area (nothing ever written here) |
| [in,out] | plain_length | plaintext data length (0 always written here) |

### ◆ AES128GCM_EncryptInit

fsp_err_t(* sce_api_t::AES128GCM_EncryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES128GCM encryption.

**Implemented as**

- R_SCE_AES128GCM_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

### ◆ AES128GCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES128GCM_EncryptUpdate) (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)

Update AES128GCM encryption.

**Implemented as**

- R_SCE_AES128GCM_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_data_length | plaintext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

### ◆ AES128GCM_EncryptFinal

fsp_err_t(* sce_api_t::AES128GCM_EncryptFinal) (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)

Finalize AES128GCM encryption.

**Implemented as**

- R_SCE_AES128GCM_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area (cipher_data_length byte) |
| [in,out] | cipher_data_length | ciphertext data length (0 always written here) |
| [in,out] | atag | authentication tag area |

### ◆ AES128GCM_DecryptInit

fsp_err_t(* sce_api_t::AES128GCM_DecryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES128GCM decryption.

**Implemented as**

- R_SCE_AES128GCM_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

### ◆ AES128GCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES128GCM_DecryptUpdate) (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)

Update AES128GCM decryption.

**Implemented as**

- R_SCE_AES128GCM_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in] | plain | plaintext data area |
| [in] | cipher_data_length | ciphertext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

### ◆ AES128GCM_DecryptFinal

fsp_err_t(* sce_api_t::AES128GCM_DecryptFinal) (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)

Finalize AES128GCM decryption.

**Implemented as**

- R_SCE_AES128GCM_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | plain | plaintext data area (cipher_data_length byte) |
| [in,out] | plain_data_length | plaintext data length (0 always written here) |
| [in,out] | atag | authentication tag area (atag_length byte) |
| [in] | atag_length | authentication tag length (4,8,12,13,14,15,16 bytes) |

### ◆ AES256GCM_EncryptInit

fsp_err_t(* sce_api_t::AES256GCM_EncryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES256GCM encryption.

**Implemented as**

- R_SCE_AES256GCM_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

### ◆ AES256GCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES256GCM_EncryptUpdate) (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)

Update AES256GCM encryption.

**Implemented as**

- R_SCE_AES256GCM_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_data_length | plaintext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

### ◆ AES256GCM_EncryptFinal

fsp_err_t(* sce_api_t::AES256GCM_EncryptFinal) (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)

Finalize AES256GCM encryption.

**Implemented as**

- R_SCE_AES256GCM_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area (cipher_data_length byte) |
| [in,out] | cipher_data_length | ciphertext data length (0 always written here) |
| [in,out] | atag | authentication tag area |

### ◆ AES256GCM_DecryptInit

fsp_err_t(* sce_api_t::AES256GCM_DecryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES256GCM decryption.

**Implemented as**

- R_SCE_AES256GCM_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | initial_vector | initialization vector area (initial_vector_length byte) |
| [in] | initial_vector_length | initialization vector length (1 ore more bytes) |

### ◆ AES256GCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES256GCM_DecryptUpdate) (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)

Update AES256GCM decryption.

**Implemented as**

- R_SCE_AES256GCM_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in] | plain | plaintext data area |
| [in] | cipher_data_length | ciphertext data length (0 or more bytes) |
| [in] | aad | additional authentication data (aad_length byte) |
| [in] | aad_length | additional authentication data length (0 or more bytes) |

### ◆ AES256GCM_DecryptFinal

fsp_err_t(* sce_api_t::AES256GCM_DecryptFinal) (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)

Finalize AES256GCM decryption.

**Implemented as**

- R_SCE_AES256GCM_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-GCM handler (work area) |
| [in,out] | plain | plaintext data area (cipher_data_length byte) |
| [in,out] | plain_data_length | plaintext data length (0 always written here) |
| [in,out] | atag | authentication tag area (atag_length byte) |
| [in] | atag_length | authentication tag length (4,8,12,13,14,15,16 bytes) |

### ◆ AES128CCM_EncryptInit

fsp_err_t(* sce_api_t::AES128CCM_EncryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES128CCM encryption.

**Implemented as**

- R_SCE_AES128CCM_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES128CCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES128CCM_EncryptUpdate) (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128CCM encryption.

**Implemented as**

- R_SCE_AES128CCM_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_length | plaintext data length |

### ◆ AES128CCM_EncryptFinal

fsp_err_t(* sce_api_t::AES128CCM_EncryptFinal) (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)

Finalize AES128CCM encryption.

**Implemented as**

- R_SCE_AES128CCM_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in,out] | cipher_length | ciphertext data length |
| [in,out] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES128CCM_DecryptInit

fsp_err_t(* sce_api_t::AES128CCM_DecryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES128CCM decryption.

**Implemented as**

- R_SCE_AES128CCM_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES128CCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES128CCM_DecryptUpdate) (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128CCM decryption.

**Implemented as**

- R_SCE_AES128CCM_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in] | cipher_length | ciphertext data length |

### ◆ AES128CCM_DecryptFinal

fsp_err_t(* sce_api_t::AES128CCM_DecryptFinal) (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)

Finalize AES128CCM decryption.

**Implemented as**

- R_SCE_AES128CCM_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | plain | plaintext data area |
| [in,out] | plain_length | plaintext data length |
| [in] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES256CCM_EncryptInit

fsp_err_t(* sce_api_t::AES256CCM_EncryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES256CCM encryption.

**Implemented as**

- R_SCE_AES256CCM_EncryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES256CCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES256CCM_EncryptUpdate) (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256CCM encryption.

**Implemented as**

- R_SCE_AES256CCM_EncryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | plain | plaintext data area |
| [in,out] | cipher | ciphertext data area |
| [in] | plain_length | plaintext data length |

### ◆ AES256CCM_EncryptFinal

fsp_err_t(* sce_api_t::AES256CCM_EncryptFinal) (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)

Finalize AES256CCM encryption.

**Implemented as**

- R_SCE_AES256CCM_EncryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | cipher | ciphertext data area |
| [in,out] | cipher_length | ciphertext data length |
| [in,out] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES256CCM_DecryptInit

fsp_err_t(* sce_api_t::AES256CCM_DecryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES256CCM decryption.

**Implemented as**

- R_SCE_AES256CCM_DecryptInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |
| [in] | nonce | Nonce |
| [in] | nonce_length | Nonce data length (7 to 13 bytes) |
| [in] | adata | additional authentication data |
| [in] | a_length | additional authentication data length (0 to 110 bytes) |
| [in] | payload_length | Payload length (any number of bytes) |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES256CCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES256CCM_DecryptUpdate) (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256CCM decryption.

**Implemented as**

- R_SCE_AES256CCM_DecryptUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in] | cipher | ciphertext data area |
| [in,out] | plain | plaintext data area |
| [in] | cipher_length | ciphertext data length |

### ◆ AES256CCM_DecryptFinal

fsp_err_t(* sce_api_t::AES256CCM_DecryptFinal) (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)

Finalize AES256CCM decryption.

**Implemented as**

- R_SCE_AES256CCM_DecryptFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CCM handler (work area) |
| [in,out] | plain | plaintext data area |
| [in,out] | plain_length | plaintext data length |
| [in] | mac | MAC area |
| [in] | mac_length | MAC length (4, 6, 8, 10, 12, 14, or 16 bytes) |

### ◆ AES128CMAC_GenerateInit

fsp_err_t(* sce_api_t::AES128CMAC_GenerateInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128CMAC generation.

**Implemented as**

- R_SCE_AES128CMAC_GenerateInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES128CMAC_GenerateUpdate

fsp_err_t(* sce_api_t::AES128CMAC_GenerateUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES128CMAC generation.

**Implemented as**

- R_SCE_AES128CMAC_GenerateUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

### ◆ AES128CMAC_GenerateFinal

fsp_err_t(* sce_api_t::AES128CMAC_GenerateFinal) (sce_cmac_handle_t *handle, uint8_t *mac)

Finalize AES128CMAC generation.

**Implemented as**

- R_SCE_AES128CMAC_GenerateFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (16byte) |

### ◆ AES128CMAC_VerifyInit

fsp_err_t(* sce_api_t::AES128CMAC_VerifyInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128CMAC verification.

**Implemented as**

- R_SCE_AES128CMAC_VerifyInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES128CMAC_VerifyUpdate

fsp_err_t(* sce_api_t::AES128CMAC_VerifyUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES128CMAC verification.

**Implemented as**

- R_SCE_AES128CMAC_VerifyUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

### ◆ AES128CMAC_VerifyFinal

fsp_err_t(* sce_api_t::AES128CMAC_VerifyFinal) (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize AES128CMAC verification.

**Implemented as**

- R_SCE_AES128CMAC_VerifyFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (mac_length byte) |
| [in,out] | mac_length | MAC data length (2 to 16 bytes) |

### ◆ AES256CMAC_GenerateInit

fsp_err_t(* sce_api_t::AES256CMAC_GenerateInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256CMAC generation.

**Implemented as**

- R_SCE_AES256CMAC_GenerateInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES256CMAC_GenerateUpdate

fsp_err_t(* sce_api_t::AES256CMAC_GenerateUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES256CMAC generation.

**Implemented as**

- R_SCE_AES256CMAC_GenerateUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

### ◆ AES256CMAC_GenerateFinal

fsp_err_t(* sce_api_t::AES256CMAC_GenerateFinal) (sce_cmac_handle_t *handle, uint8_t *mac)

Finalize AES256CMAC generation.

**Implemented as**

- R_SCE_AES256CMAC_GenerateFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (16byte) |

### ◆ AES256CMAC_VerifyInit

fsp_err_t(* sce_api_t::AES256CMAC_VerifyInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256CMAC verification.

**Implemented as**

- R_SCE_AES256CMAC_VerifyInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | wrapped_key | 256-bit AES wrapped key |

### ◆ AES256CMAC_VerifyUpdate

fsp_err_t(* sce_api_t::AES256CMAC_VerifyUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES256CMAC verification.

**Implemented as**

- R_SCE_AES256CMAC_VerifyUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in] | message | message data area (message_length byte) |
| [in] | message_length | message data length (0 or more bytes) |

### ◆ AES256CMAC_VerifyFinal

fsp_err_t(* sce_api_t::AES256CMAC_VerifyFinal) (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize AES256CMAC verification.

**Implemented as**

- R_SCE_AES256CMAC_VerifyFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | AES-CMAC handler (work area) |
| [in,out] | mac | MAC data area (mac_length byte) |
| [in,out] | mac_length | MAC data length (2 to 16 bytes) |

### ◆ SHA256_Init

fsp_err_t(* sce_api_t::SHA256_Init) (sce_sha_md5_handle_t *handle)

Initialize SHA-256 Calculation.

**Implemented as**

- R_SCE_SHA256_Init()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |

### ◆ SHA256_Update

fsp_err_t(* sce_api_t::SHA256_Update) (sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length)

Update SHA-256 Calculation.

**Implemented as**

- R_SCE_SHA256_Update()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |
| [in] | message | message data area |
| [in] | message_length | message data length |

### ◆ SHA256_Final

fsp_err_t(* sce_api_t::SHA256_Final) (sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length)

Finalize SHA-256 Calculation.

**Implemented as**

- R_SCE_SHA256_Final()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA handler (work area) |
| [in,out] | digest | hasha data area |
| [in,out] | digest_length | hash data length (32bytes) |

### ◆ RSA1024_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::RSA1024_WrappedKeyPairGenerate) (sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key)

This API outputs 1024-bit RSA wrapped pair key.

**Implemented as**

- R_SCE_RSA1024_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ RSA2048_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::RSA2048_WrappedKeyPairGenerate) (sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key)

This API outputs 2048-bit RSA wrapped pair key.

**Implemented as**

- R_SCE_RSA2048_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ RSA1024_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA1024_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key)

This API outputs 1024-bit RSA public wrapped key.

**Implemented as**

- R_SCE_RSA1024_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA public wrapped key |

### ◆ RSA1024_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::RSA1024_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key)

This API outputs 1024-bit RSA private wrapped key.

**Implemented as**

- R_SCE_RSA1024_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 1024-bit RSA private wrapped key |

### ◆ RSA2048_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA2048_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key)

This API outputs 2048-bit RSA public wrapped key.

**Implemented as**

- R_SCE_RSA2048_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA public wrapped key |

### ◆ RSA2048_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::RSA2048_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key)

This API outputs 2048-bit RSA private wrapped key.

**Implemented as**

- R_SCE_RSA2048_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA private wrapped key |

### ◆ RSA3072_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA3072_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key)

This API outputs 3072-bit RSA public wrapped key.

**Implemented as**

- R_SCE_RSA3072_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 3072-bit RSA public wrapped key |

### ◆ RSA4096_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA4096_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key)

This API outputs 4096-bit RSA public wrapped key.

**Implemented as**

- R_SCE_RSA4096_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 4096-bit RSA public wrapped key |

### ◆ RSASSA_PKCS1024_SignatureGenerate

fsp_err_t(* sce_api_t::RSASSA_PKCS1024_SignatureGenerate) (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature generation.

**Implemented as**

- R_SCE_RSASSA_PKCS1024_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Inputs the 1024-bit RSA private wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSASSA_PKCS2048_SignatureGenerate

fsp_err_t(* sce_api_t::RSASSA_PKCS2048_SignatureGenerate) (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature generation.

**Implemented as**

- R_SCE_RSASSA_PKCS2048_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Inputs the 2048-bit RSA private wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSASSA_PKCS1024_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS1024_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

**Implemented as**

- R_SCE_RSASSA_PKCS1024_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify |
| [in] | message_hash | Message text or hash value to verify |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSASSA_PKCS2048_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS2048_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

**Implemented as**

- R_SCE_RSASSA_PKCS2048_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify |
| [in] | message_hash | Message text or hash value to verify |
| [in] | wrapped_key | Inputs the 2048-bit RSA public wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSASSA_PKCS3072_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS3072_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

**Implemented as**

- R_SCE_RSASSA_PKCS3072_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify |
| [in] | message_hash | Message text or hash value to verify |
| [in] | wrapped_key | Inputs the 3072-bit RSA public wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSASSA_PKCS4096_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS4096_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

**Implemented as**

- R_SCE_RSASSA_PKCS4096_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to verify |
| [in] | message_hash | Message text or hash value to verify |
| [in] | wrapped_key | Inputs the 4096-bit RSA public wrapped key. |
| [in] | hash_type | Only HW_SCE_RSA_HASH_SHA256 is supported |

### ◆ RSAES_PKCS1024_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS1024_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

**Implemented as**

- R_SCE_RSAES_PKCS1024_Encrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext |
| [in,out] | cipher | ciphertext |
| [in] | wrapped_key | Inputs the 1024-bit RSA public wrapped key. |

### ◆ RSAES_PKCS2048_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS2048_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

**Implemented as**

- R_SCE_RSAES_PKCS2048_Encrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext |
| [in,out] | cipher | ciphertext |
| [in] | wrapped_key | Inputs the 2048-bit RSA public wrapped key. |

### ◆ RSAES_PKCS3072_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS3072_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

**Implemented as**

- R_SCE_RSAES_PKCS3072_Encrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext |
| [in,out] | cipher | ciphertext |
| [in] | wrapped_key | Inputs the 3072-bit RSA public wrapped key. |

### ◆ RSAES_PKCS4096_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS4096_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

**Implemented as**

- R_SCE_RSAES_PKCS4096_Encrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | plain | plaintext |
| [in,out] | cipher | ciphertext |
| [in] | wrapped_key | Inputs the 4096-bit RSA public wrapped key. |

### ◆ RSAES_PKCS1024_Decrypt

fsp_err_t(* sce_api_t::RSAES_PKCS1024_Decrypt) (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 decryption.

**Implemented as**

- R_SCE_RSAES_PKCS1024_Decrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | cipher | ciphertext |
| [in,out] | plain | plaintext |
| [in] | wrapped_key | Inputs the 1024-bit RSA private wrapped key. |

### ◆ RSAES_PKCS2048_Decrypt

fsp_err_t(* sce_api_t::RSAES_PKCS2048_Decrypt) (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 decryption.

**Implemented as**

- R_SCE_RSAES_PKCS2048_Decrypt()

**Parameters**

| | | |
|---|---|---|
| [in] | cipher | ciphertext |
| [in,out] | plain | plaintext |
| [in] | wrapped_key | Inputs the 2048-bit RSA private wrapped key. |

### ◆ SHA256HMAC_EncryptedKeyWrap

fsp_err_t(* sce_api_t::SHA256HMAC_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key)

This API outputs HMAC-SHA256 wrapped key.

**Implemented as**

- R_SCE_SHA256HMAC_EncryptedKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | HMAC-SHA256 wrapped key |

### ◆ SHA256HMAC_GenerateInit

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateInit) (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)

Initialize HMAC-SHA256 generation.

**Implemented as**

- R_SCE_SHA256HMAC_GenerateInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handler (work area) |
| [in] | wrapped_key | MAC wrapped key |

### ◆ SHA256HMAC_GenerateUpdate

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateUpdate) (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)

Update HMAC-SHA256 generation.

**Implemented as**

- R_SCE_SHA256HMAC_GenerateUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | message | Message area |
| [in] | message_length | Message length |

### ◆ SHA256HMAC_GenerateFinal

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateFinal) (sce_hmac_sha_handle_t *handle, uint8_t *mac)

Finalize HMAC-SHA256 generation.

**Implemented as**

- R_SCE_SHA256HMAC_GenerateFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in,out] | mac | HMAC area (32 bytes) |

### ◆ SHA256HMAC_VerifyInit

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyInit) (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)

Initialize HMAC-SHA256 verification.

**Implemented as**

- R_SCE_SHA256HMAC_VerifyInit()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handler (work area) |
| [in] | wrapped_key | MAC wrapped key |

### ◆ SHA256HMAC_VerifyUpdate

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyUpdate) (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)

Update HMAC-SHA256 verification.

**Implemented as**

- R_SCE_SHA256HMAC_VerifyUpdate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | message | Message area |
| [in] | message_length | Message length |

### ◆ SHA256HMAC_VerifyFinal

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyFinal) (sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize HMAC-SHA256 verification.

**Implemented as**

- R_SCE_SHA256HMAC_VerifyFinal()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | SHA-HMAC handle (work area) |
| [in] | mac | HMAC area |
| [in] | mac_length | HMAC length |

### ◆ ECC_secp192r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp192r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp192r1 wrapped pair key.

**Implemented as**

- R_SCE_ECC_secp192r1_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp192r1 public key and private key pair |

### ◆ ECC_secp224r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp224r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp224r1 wrapped pair key.

**Implemented as**

- R_SCE_ECC_secp224r1_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp224r1 public key and private key pair |

### ◆ ECC_secp256r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp256r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp256r1 wrapped pair key.

**Implemented as**

- R_SCE_ECC_secp256r1_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp256r1 public key and private key pair |

### ◆ ECC_secp384r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp384r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp384r1 wrapped pair key.

**Implemented as**

- R_SCE_ECC_secp384r1_WrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | wrapped_pair_key | Wrapped pair key for secp384r1 public key and private key pair |

### ◆ ECC_secp192r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp192r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp192r1 public wrapped key.

**Implemented as**

- R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp192r1 public wrapped key |

### ◆ ECC_secp224r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp224r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp224r1 public wrapped key.

**Implemented as**

- R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp224r1 public wrapped key |

### ◆ ECC_secp256r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp256r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp256r1 public wrapped key.

**Implemented as**

- R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp256r1 public wrapped key |

### ◆ ECC_secp384r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp384r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp384r1 public wrapped key.

**Implemented as**

- R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp384r1 public wrapped key |

### ◆ ECC_secp192r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp192r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp192r1 private wrapped key.

**Implemented as**

- R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp192r1 private wrapped key |

### ◆ ECC_secp224r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp224r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp224r1 private wrapped key.

**Implemented as**

- R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp224r1 private wrapped key |

### ◆ ECC_secp256r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp256r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp256r1 private wrapped key.

**Implemented as**

- R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp256r1 private wrapped key |

### ◆ ECC_secp384r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp384r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp384r1 private wrapped key.

**Implemented as**

- R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap()

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | secp384r1 private wrapped key |

### ◆ ECDSA_secp192r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp192r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

**Implemented as**

- R_SCE_ECDSA_secp192r1_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Input wrapped key of secp192r1 private key. |

### ◆ ECDSA_secp224r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp224r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

**Implemented as**

- R_SCE_ECDSA_secp224r1_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Input wrapped key of secp224r1 private key. |

### ◆ ECDSA_secp256r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp256r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

**Implemented as**

- R_SCE_ECDSA_secp256r1_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Input wrapped key of secp256r1 private key. |

### ◆ ECDSA_secp384r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp384r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

**Implemented as**

- R_SCE_ECDSA_secp384r1_SignatureGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | message_hash | Message or hash value to which to attach signature |
| [in,out] | signature | Signature text storage destination information |
| [in] | wrapped_key | Input wrapped key of secp384r1 private key. |

### ◆ ECDSA_secp192r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp192r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

**Implemented as**

- R_SCE_ECDSA_secp192r1_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified |
| [in,out] | message_hash | Message or hash value to be verified |
| [in] | wrapped_key | Input wrapped key of secp192r1 public key. |

### ◆ ECDSA_secp224r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp224r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

**Implemented as**

- R_SCE_ECDSA_secp224r1_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified |
| [in,out] | message_hash | Message or hash value to be verified |
| [in] | wrapped_key | Input wrapped key of secp224r1 public key. |

### ◆ ECDSA_secp256r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp256r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

**Implemented as**

- R_SCE_ECDSA_secp256r1_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified |
| [in,out] | message_hash | Message or hash value to be verified |
| [in] | wrapped_key | Input wrapped key of secp256r1 public key. |

### ◆ ECDSA_secp384r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp384r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

**Implemented as**

- R_SCE_ECDSA_secp384r1_SignatureVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | signature | Signature text information to be verified |
| [in,out] | message_hash | Message or hash value to be verified |
| [in] | wrapped_key | Input wrapped key of secp384r1 public key. |

### ◆ ECDH_secp256r1_Init

fsp_err_t(* sce_api_t::ECDH_secp256r1_Init) (sce_ecdh_handle_t *handle, uint32_t key_type, uint32_t use_key_id)

secp256r1 ECDH Initialization.

**Implemented as**

- R_SCE_ECDH_secp256r1_Init()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | key_type | Key exchange type |
| [in] | use_key_id | use key_id or not |

### ◆ ECDH_secp256r1_PublicKeySign

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeySign) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

secp256r1 ECDH public key Signature.

**Implemented as**

- R_SCE_ECDH_secp256r1_PublicKeySign()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | ecc_public_wrapped_key | For ECDHE, input a null pointer. For ECDH, input the wrapped key of a secp256r1 public key. |
| [in] | ecc_private_wrapped_key | secp256r1 private key for signature generation |
| [in,out] | public_key | User secp256r1 public key (512-bit) for key exchange. |
| [in,out] | signature | Signature text storage destination information |
| [in,out] | wrapped_key | For ECDHE, a private wrapped key generated from a random number. Not output for ECDH. |

### ◆ ECDH_secp256r1_PublicKeyVerify

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeyVerify) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key)

secp256r1 ECDH public key verification.

**Implemented as**

- R_SCE_ECDH_secp256r1_PublicKeyVerify()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | ecc_public_wrapped_key | Public wrapped key area for signature verification |
| [in] | public_key_data | secp256r1 public key (512-bit) |
| [in] | signature | ECDSA secp256r1 signature of ecc_public_wrapped_key |
| [in,out] | wrapped_key | wrapped key of ecc_public_wrapped_key |

### ◆ ECDH_secp256r1_PublicKeyReadWithoutSignature

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeyReadWithoutSignature) (sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key)

Output the key index of QeU without signature verification.

**Implemented as**

- R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | public_key_data | secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) \|\| public key (512-bit) |
| [in,out] | wrapped_key | wrapped key of ecc_public_wrapped_key |

### ◆ ECDH_secp256r1_SharedSecretCalculate

fsp_err_t(* sce_api_t::ECDH_secp256r1_SharedSecretCalculate) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key)

secp256r1 ECDH shared secret calculation.

**Implemented as**

- R_SCE_ECDH_secp256r1_SharedSecretCalculate()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | ecc_public_wrapped_key | Public wrapped key |
| [in] | ecc_private_wrapped_key | Private wrapped key |
| [in,out] | shared_secret_wrapped_key | Wrapped key of shared secret Z calculated by ECDH key exchange |

### ◆ ECDH_secp256r1_KeyDerivation

fsp_err_t(* sce_api_t::ECDH_secp256r1_KeyDerivation) (sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key)

secp256r1 ECDH key derivation.

**Implemented as**

- R_SCE_ECDH_secp256r1_KeyDerivation()

**Parameters**

| | | |
|---|---|---|
| [in,out] | handle | ECDH handler (work area) |
| [in] | shared_secret_wrapped_key | Z wrapped key calculated by R_SCE_ECDH_secp256r1_SharedSecretCalculate |
| [in] | key_type | Derived key type |
| [in] | kdf_type | Algorithm used for key derivation calculation |
| [in] | other_info | Additional data used for key derivation calculation |
| [in] | other_info_length | Data length of other_info |
| [in] | salt_wrapped_key | Salt wrapped key |
| [in,out] | wrapped_key | Wrapped key corresponding to key_type. |

### ◆ TLS_RootCertificateRSA2048PublicKeyInstall

fsp_err_t(* sce_api_t::TLS_RootCertificateRSA2048PublicKeyInstall) (uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key)

Generate TLS RSA Public key index data

**Implemented as**

- R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall()

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_provisioning_key | the provisioning key includes encrypted CBC/CBC-MAC key for user key |
| [in] | initial_vector | the initial_vector for user key CBC encrypt |
| [in] | encrypted_key | the user key encrypted with AES128-ECB mode |
| [out] | wrapped_key | the user Key Generation Information (141 words) of RSA2048 bit |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate

fsp_err_t(* sce_api_t::TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate) (sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key)

Generate TLS ECC key pair

**Implemented as**

- R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | tls_p256_ecc_wrapped_key | P256 ECC key index for TLS |
| [in] | ephemeral_ecdh_public_key | ephemeral ECDH public key |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_RootCertificateVerify

fsp_err_t(* sce_api_t::TLS_RootCertificateVerify) (uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key)

Verify root CA certificate.

**Implemented as**

- R_SCE_TLS_RootCertificateVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | certificate | certificates. |
| [in] | certificate_length | byte size of certificates. |
| [in] | public_key_n_start_position | start position of public key n. |
| [in] | public_key_n_end_position | end position of public key n. |
| [in] | public_key_e_start_position | start position of public key e. |
| [in] | public_key_e_end_position | end position of public key e. |
| [in] | signature | signature for certificates. |
| [out] | encrypted_root_public_key | public key for RSA 2048bit. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_CertificateVerify

fsp_err_t(* sce_api_t::TLS_CertificateVerify) (uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key)

Verify server certificate and intermediate certificate.

**Implemented as**

- R_SCE_TLS_CertificateVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | input_public_key | public key. |
| [in] | certificate | certificates. |
| [in] | certificate_length | byte size of certificates. |
| [in] | signature | signature for certificates. |
| [in] | public_key_n_start_position | start position of public key n. |
| [in] | public_key_n_end_position | end position of public key n. |
| [in] | public_key_e_start_position | start position of public key e. |
| [in] | public_key_e_end_position | end position of public key e. |
| [out] | encrypted_output_public_key | public key for RSA 2048bit. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_PreMasterSecretGenerateForRSA2048

fsp_err_t(* sce_api_t::TLS_PreMasterSecretGenerateForRSA2048) (uint32_t *sce_pre_master_secret)

Generate encrypted pre-master secret.

**Implemented as**

- R_SCE_TLS_PreMasterSecretGenerateForRSA2048()

**Parameters**

| | | |
|---|---|---|
| [out] | sce_pre_master_secret | pre-master secret value for SCE. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |

### ◆ TLS_MasterSecretGenerate

fsp_err_t(* sce_api_t::TLS_MasterSecretGenerate) (uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret)

Generate encrypted master secret.

**Implemented as**

- R_SCE_TLS_MasterSecretGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | select_cipher_suite | cipher suite type |
| [in] | sce_pre_master_secret | pre-master secret value for SCE. |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [out] | sce_master_secret | master secret value with SCE-specific conversion. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_PreMasterSecretEncryptWithRSA2048

fsp_err_t(* sce_api_t::TLS_PreMasterSecretEncryptWithRSA2048) (uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret)

Output the result encrypted pre-master secret with RSA 2048bit

**Implemented as**

- R_SCE_TLS_PreMasterSecretEncryptWithRSA2048()

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_public_key | public key data. |
| [in] | sce_pre_master_secret | pre-master secret value. |
| [out] | encrypted_pre_master_secret | the value encrypted pre-master secret. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_SessionKeyGenerate

fsp_err_t(* sce_api_t::TLS_SessionKeyGenerate) (uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector)

Output various key information.

**Implemented as**

- R_SCE_TLS_SessionKeyGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | select_cipher_suite | Key suite information number. |
| [in] | sce_master_secret | master secret value. |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [in] | nonce_explicit | nonce value |

| | | |
|---|---|---|
| [out] | client_mac_wrapped_key | the mac key during communication from client to server. |
| [out] | server_mac_wrapped_key | the mac key during communication from server to client. |
| [out] | client_crypto_wrapped_key | the crypto key during communication from client to server. |
| [out] | server_crypto_wrapped_key | the crypto key during communication from server to client. |
| [in] | client_initial_vector | not use. |
| [in] | server_initial_vector | not use. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_VerifyDataGenerate

fsp_err_t(* sce_api_t::TLS_VerifyDataGenerate) (uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data)

Generate verify data.

**Implemented as**

- R_SCE_TLS_VerifyDataGenerate()

**Parameters**

| | | |
|---|---|---|
| [in] | select_verify_data | Select Client/Server data. |
| [in] | sce_master_secret | master secret data. |
| [in] | hand_shake_hash | TLS hand shake message SHA256 HASH value. |
| [out] | verify_data | verify data. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_ServerKeyExchangeVerify

fsp_err_t(* sce_api_t::TLS_ServerKeyExchangeVerify) (uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key)

Retrives ECDH public key.

**Implemented as**

- R_SCE_TLS_ServerKeyExchangeVerify()

**Parameters**

| | | |
|---|---|---|
| [in] | public_key_type | key type |
| [in] | client_random | random value reported ClientHello. |
| [in] | server_random | random value reported ServerHello. |
| [in] | server_ephemeral_ecdh_public_key | Ephemeral ECDH public key from Server. |
| [in] | server_key_exchange_siguna ture | Server Key Exchange sigunature. |
| [in] | encrypted_public_key | encrypted public key. |
| [out] | encrypted_ephemeral_ecdh_ public_key | encrypted Ephemeral ECDH public key. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLIC T | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ TLS_PreMasterSecretGenerateForECC_secp256r1

fsp_err_t(* sce_api_t::TLS_PreMasterSecretGenerateForECC_secp256r1) (uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret)

Generate encrypted pre-master secret.

**Implemented as**

- R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1()

**Parameters**

| | | |
|---|---|---|
| [in] | encrypted_public_key | encrypted public key |
| [in] | tls_p256_ecc_wrapped_key | P-256 ECC key index. |
| [out] | sce_pre_master_secret | encrypted pre-master secret value for SCE. |

**Return values**

| | |
|---|---|
| FSP_SUCCESS | Normal termination |
| FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT | A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine. |
| FSP_ERR_CRYPTO_SCE_FAIL | An internal error occurred. |

### ◆ sce_instance_t

struct sce_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| sce_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| sce_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| sce_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ sce_rsa_byte_data_t

typedef sce_byte_data_t sce_rsa_byte_data_t

byte data

RSA byte data structure

### ◆ sce_ecdsa_byte_data_t

| typedef sce_byte_data_t sce_ecdsa_byte_data_t |
|---|
| byte data<br>ECDSA byte data structure |

### ◆ sce_ctrl_t

| typedef void sce_ctrl_t |
|---|
| SCE Control block. Allocate an instance specific control block to pass into the API calls.<br><br>**Implemented as**<br>◦ sce_instance_ctrl_t |

## 5.3.13.4 SCE key injection Interface

Interfaces » Security

### Detailed Description

Interface for key injection by Secure Crypto Engine (SCE) functions.

# Summary

The SCE key injection interface provides SCE functionality.

### Data Structures

| | |
|---|---|
| struct | sce_key_injection_api_t |

### Data Structure Documentation

### ◆ sce_key_injection_api_t

| struct sce_key_injection_api_t | |
|---|---|
| Functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | AES128_InitialKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |

| | |
|---|---|
| fsp_err_t(* | AES192_InitialKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | AES256_InitialKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | KeyUpdateKeyWrap )(const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_key_update_key_t *const key_update_key) |
| | |
| fsp_err_t(* | AES128_EncryptedKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | AES192_EncryptedKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | AES256_EncryptedKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA3072_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa3072_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA4096_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, |

| | |
|---|---|
| | sce_rsa4096_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_InitialPrivateKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | RSA2048_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256k1_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_InitialPrivateKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256k1_InitialPrivateKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |

| | |
|---|---|
| | |
| fsp_err_t(* | ECC_secp384r1_InitialPrivateKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256k1_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256r1_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp256k1_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_secp384r1_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP256r1_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP256r1_InitialPrivateKeyWrap )(const uint8_t *const |

| | |
|---|---|
| | key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP384r1_InitialPublicKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP384r1_InitialPrivateKeyWrap )(const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP256r1_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP256r1_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP384r1_EncryptedPublicKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key) |
| | |
| fsp_err_t(* | ECC_brainpoolP384r1_EncryptedPrivateKeyWrap )(const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key) |
| | |

## Field Documentation

### ◆ AES128_InitialKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES128_InitialKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key)

This API outputs 128-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES192_InitialKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES192_InitialKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key)

This API outputs 192-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 192-bit AES wrapped key |

### ◆ AES256_InitialKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES256_InitialKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_aes_wrapped_key_t *const wrapped_key)

This API outputs 256-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

### ◆ KeyUpdateKeyWrap

fsp_err_t(* sce_key_injection_api_t::KeyUpdateKeyWrap) (const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_key_update_key_t *const key_update_key)

This API outputs key update key.

**Parameters**

| | | |
|---|---|---|
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | key_update_key | Key update key |

### ◆ AES128_EncryptedKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES128_EncryptedKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key)

This API updates 128-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 128-bit AES wrapped key |

### ◆ AES192_EncryptedKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES192_EncryptedKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key)

This API updates 192-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 192-bit AES wrapped key |

### ◆ AES256_EncryptedKeyWrap

fsp_err_t(* sce_key_injection_api_t::AES256_EncryptedKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_aes_wrapped_key_t *const wrapped_key)

This API outputs 256-bit AES wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit AES wrapped key |

### ◆ RSA2048_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA2048_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key)

This API outputs 2048-bit RSA public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

### ◆ RSA3072_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA3072_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa3072_public_wrapped_key_t *const wrapped_key)

This API outputs 3072-bit RSA public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 3072-bit RSA wrapped key |

### ◆ RSA4096_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA4096_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa4096_public_wrapped_key_t *const wrapped_key)

This API outputs 4096-bit RSA public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 4096-bit RSA wrapped key |

### ◆ RSA2048_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA2048_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key)

This API outputs 2048-bit RSA private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

### ◆ RSA2048_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA2048_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_public_wrapped_key_t *const wrapped_key)

This API outputs 2048-bit RSA public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 2048-bit RSA wrapped key |

### ◆ RSA2048_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::RSA2048_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_rsa2048_private_wrapped_key_t *const wrapped_key)

This API outputs 2048-bit RSA private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit RSA wrapped key |

### ◆ ECC_secp256r1_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256r1_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp256k1_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256k1_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp384r1_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp384r1_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_secp256r1_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256r1_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp256k1_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256k1_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp384r1_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp384r1_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_secp256r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256r1_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp256k1_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256k1_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp384r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp384r1_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_secp256r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256r1_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp256k1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp256k1_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_secp384r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_secp384r1_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_brainpoolP256r1_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP256r1_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit Brainpool ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_brainpoolP256r1_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP256r1_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit Brainpool ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_brainpoolP384r1_InitialPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP384r1_InitialPublicKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 384-bit Brainpool ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_brainpoolP384r1_InitialPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP384r1_InitialPrivateKeyWrap) (const uint8_t *const key_type, const uint8_t *const wrapped_user_factory_programming_key, const uint8_t *const initial_vector, const uint8_t *const encrypted_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 384-bit Brainpool ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | key_type | Key type whether encrypted_key or plain key |
| [in] | wrapped_user_factory_programming_key | Provisioning key wrapped by the DLM server |
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encrypted and MAC appended |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

### ◆ ECC_brainpoolP256r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP256r1_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_brainpoolP256r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP256r1_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 256-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_brainpoolP384r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP384r1_EncryptedPublicKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_public_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC public wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 256-bit ECC wrapped key |

### ◆ ECC_brainpoolP384r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_key_injection_api_t::ECC_brainpoolP384r1_EncryptedPrivateKeyWrap) (const uint8_t *const initial_vector, const uint8_t *const encrypted_key, const sce_key_update_key_t *const key_update_key, sce_ecc_private_wrapped_key_t *const wrapped_key)

This API outputs 384-bit ECC private wrapped key.

**Parameters**

| | | |
|---|---|---|
| [in] | initial_vector | Initialization vector when generating encrypted_key |
| [in] | encrypted_key | User key encryptedand MAC appended |
| [in] | key_update_key | Key update keyring |
| [in,out] | wrapped_key | 384-bit ECC wrapped key |

## 5.3.14 Sensor

Interfaces

### Detailed Description

Sensor Interfaces.

### Modules

| | |
|---|---|
| FSXXXX Middleware Interface | Interface for FSXXXX Middleware functions. |
| HS300X Middleware Interface | Interface for HS300X Middleware functions. |
| HS400X Middleware Interface | Interface for HS400X Middleware functions. |
| SENSOR Middleware Interface | Interface for AIR SENSOR Middleware functions. |
| ZMOD4XXX Middleware Interface | |

| | |
|---|---|
| | Interface for ZMOD4XXX Middleware functions. |

### 5.3.14.1 FSXXXX Middleware Interface

Interfaces » Sensor

#### Detailed Description

Interface for FSXXXX Middleware functions.

## Summary

The FSXXXX interface provides FSXXXX functionality.

#### Data Structures

| | |
|---|---|
| struct | rm_fsxxxx_callback_args_t |
| struct | rm_fsxxxx_raw_data_t |
| struct | rm_fsxxxx_sensor_data_t |
| struct | rm_fsxxxx_data_t |
| struct | rm_fsxxxx_cfg_t |
| struct | rm_fsxxxx_api_t |
| struct | rm_fsxxxx_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | rm_fsxxxx_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | rm_fsxxxx_event_t |

#### Data Structure Documentation

◆ **rm_fsxxxx_callback_args_t**

| |
|---|
| struct rm_fsxxxx_callback_args_t |
| FSXXXX callback parameter definition |

◆ **rm_fsxxxx_raw_data_t**

| struct rm_fsxxxx_raw_data_t |
| --- |
| FSXXXX raw data |

### ◆ rm_fsxxxx_sensor_data_t

| struct rm_fsxxxx_sensor_data_t | | |
| --- | --- | --- |
| FSXXXX sensor data block | | |
| Data Fields | | |
| int16_t | integer_part | |
| int16_t | decimal_part | To two decimal places. |

### ◆ rm_fsxxxx_data_t

| struct rm_fsxxxx_data_t |
| --- |
| FSXXXX data block |

### ◆ rm_fsxxxx_cfg_t

| struct rm_fsxxxx_cfg_t | |
| --- | --- |
| FSXXXX Configuration | |
| **Data Fields** | |
| rm_comms_instance_t const * | p_instance |
| | Pointer to Communications Middleware instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void(* | p_callback )(rm_fsxxxx_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |

### ◆ rm_fsxxxx_api_t

| |
| --- |

| struct rm_fsxxxx_api_t | |
|---|---|
| FSXXXX APIs | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | read )(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data) |
| | |
| fsp_err_t(* | dataCalculate )(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fsxxxx_data) |
| | |
| fsp_err_t(* | close )(rm_fsxxxx_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_fsxxxx_api_t::open) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_cfg_t const *const p_cfg)

Open sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ read

fsp_err_t(* rm_fsxxxx_api_t::read) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data)

Read ADC data from FSXXXX.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data structure. |

### ◆ dataCalculate

fsp_err_t(* rm_fsxxxx_api_t::dataCalculate) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fsxxxx_data)

Calculate flow values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_fsxxxx_data | Pointer to FSXXXX data structure. |

### ◆ close

fsp_err_t(* rm_fsxxxx_api_t::close) (rm_fsxxxx_ctrl_t *const p_ctrl)

Close FSXXXX.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_fsxxxx_instance_t

struct rm_fsxxxx_instance_t

FSXXXX instance

| Data Fields | | |
|---|---|---|
| rm_fsxxxx_ctrl_t * | p_ctrl | Pointer to the control structure for this instance |
| rm_fsxxxx_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance |
| rm_fsxxxx_api_t const * | p_api | Pointer to the API structure for this instance |

## Typedef Documentation

### ◆ rm_fsxxxx_ctrl_t

typedef void rm_fsxxxx_ctrl_t

FSXXXX control block. Allocate an instance specific control block to pass into the FSXXXX API calls.

## Enumeration Type Documentation

### ◆ rm_fsxxxx_event_t

| enum rm_fsxxxx_event_t |
| --- |
| Event in the callback function |

### 5.3.14.2 HS300X Middleware Interface

Interfaces » Sensor

#### Detailed Description

Interface for HS300X Middleware functions.

# Summary

The HS300X interface provides HS300X functionality.

#### Data Structures

| | |
| --- | --- |
| struct | rm_hs300x_callback_args_t |
| struct | rm_hs300x_raw_data_t |
| struct | rm_hs300x_sensor_data_t |
| struct | rm_hs300x_data_t |
| struct | rm_hs300x_cfg_t |
| struct | rm_hs300x_api_t |
| struct | rm_hs300x_instance_t |

#### Typedefs

| | |
| --- | --- |
| typedef void | rm_hs300x_ctrl_t |

#### Enumerations

| | |
| --- | --- |
| enum | rm_hs300x_event_t |
| enum | rm_hs300x_data_type_t |
| enum | rm_hs300x_resolution_t |

#### Data Structure Documentation

### ◆ rm_hs300x_callback_args_t

| struct rm_hs300x_callback_args_t |
|---|
| HS300X callback parameter definition |

### ◆ rm_hs300x_raw_data_t

| struct rm_hs300x_raw_data_t | | |
|---|---|---|
| HS300X raw data | | |
| Data Fields | | |
| uint8_t | humidity[2] | Upper 2 bits of 0th element are data status. |
| uint8_t | temperature[2] | Lower 2 bits of 1st element are mask. |

### ◆ rm_hs300x_sensor_data_t

| struct rm_hs300x_sensor_data_t | | |
|---|---|---|
| HS300X sensor data block | | |
| Data Fields | | |
| int16_t | integer_part | |
| int16_t | decimal_part | To two decimal places. |

### ◆ rm_hs300x_data_t

| struct rm_hs300x_data_t |
|---|
| HS300X data block |

### ◆ rm_hs300x_cfg_t

| struct rm_hs300x_cfg_t | |
|---|---|
| HS300X Configuration | |
| **Data Fields** | |
| rm_comms_instance_t const * | p_instance |
| | Pointer to Communications Middleware instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |

| | |
|---|---|
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void(* | p_callback )(rm_hs300x_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |

### ◆ rm_hs300x_api_t

| struct rm_hs300x_api_t | |
|---|---|
| HS300X APIs | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | measurementStart )(rm_hs300x_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data) |
| | |
| fsp_err_t(* | dataCalculate )(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data) |
| | |
| fsp_err_t(* | programmingModeEnter )(rm_hs300x_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | resolutionChange )(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution) |
| | |
| fsp_err_t(* | sensorIdGet )(rm_hs300x_ctrl_t *const p_ctrl, uint32_t *const p_sensor_id) |
| | |
| fsp_err_t(* | programmingModeExit )(rm_hs300x_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(rm_hs300x_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_hs300x_api_t::open) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_cfg_t const *const p_cfg)

Open sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ measurementStart

fsp_err_t(* rm_hs300x_api_t::measurementStart) (rm_hs300x_ctrl_t *const p_ctrl)

Start a measurement.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* rm_hs300x_api_t::read) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data)

Read ADC data from HS300X.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data structure. |

### ◆ dataCalculate

fsp_err_t(* rm_hs300x_api_t::dataCalculate) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data)

Calculate humidity and temperature values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_hs300x_data | Pointer to HS300X data structure. |

### ◆ programmingModeEnter

fsp_err_t(* rm_hs300x_api_t::programmingModeEnter) (rm_hs300x_ctrl_t *const p_ctrl)

Enter the programming mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ resolutionChange

fsp_err_t(* rm_hs300x_api_t::resolutionChange) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution)

Change the sensor resolution.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | data_type | Data type of HS300X. |
| [in] | resolution | Resolution type of HS300X. |

### ◆ sensorIdGet

fsp_err_t(* rm_hs300x_api_t::sensorIdGet) (rm_hs300x_ctrl_t *const p_ctrl, uint32_t *const p_sensor_id)

Get the sensor ID.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_sensor_id | Pointer to sensor ID of HS300X. |

### ◆ programmingModeExit

| fsp_err_t(* rm_hs300x_api_t::programmingModeExit) (rm_hs300x_ctrl_t *const p_ctrl) |
|---|

Exit the programming mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ close

| fsp_err_t(* rm_hs300x_api_t::close) (rm_hs300x_ctrl_t *const p_ctrl) |
|---|

Close HS300X.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_hs300x_instance_t

| struct rm_hs300x_instance_t | | |
|---|---|---|
| HS300X instance | | |
| Data Fields | | |
| rm_hs300x_ctrl_t * | p_ctrl | Pointer to the control structure for this instance |
| rm_hs300x_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance |
| rm_hs300x_api_t const * | p_api | Pointer to the API structure for this instance |

## Typedef Documentation

### ◆ rm_hs300x_ctrl_t

| typedef void rm_hs300x_ctrl_t |
|---|
| HS300X control block. Allocate an instance specific control block to pass into the HS300X API calls. |

## Enumeration Type Documentation

### ◆ rm_hs300x_event_t

| enum rm_hs300x_event_t |
|---|
| Event in the callback function |

### ◆ rm_hs300x_data_type_t

| enum rm_hs300x_data_type_t |
| --- |
| Data type of HS300X |

### ◆ rm_hs300x_resolution_t

| enum rm_hs300x_resolution_t |
| --- |
| Resolution type of HS300X |

### 5.3.14.3 HS400X Middleware Interface

Interfaces » Sensor

#### Detailed Description

Interface for HS400X Middleware functions.

# Summary

The HS400X interface provides HS400X functionality.

#### Data Structures

| | |
| --- | --- |
| struct | rm_hs400x_callback_args_t |
| struct | rm_hs400x_resolutions_t |
| struct | rm_hs400x_raw_data_t |
| struct | rm_hs400x_sensor_data_t |
| struct | rm_hs400x_data_t |
| struct | rm_hs400x_cfg_t |
| struct | rm_hs400x_api_t |
| struct | rm_hs400x_instance_t |

#### Typedefs

| | |
| --- | --- |
| typedef void | rm_hs400x_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_hs400x_event_t |
| enum | rm_hs400x_temperature_resolution_t |
| enum | rm_hs400x_humidity_resolution_t |
| enum | rm_hs400x_periodic_measurement_frequency_t |

## Data Structure Documentation

### ◆ rm_hs400x_callback_args_t

| struct rm_hs400x_callback_args_t |
|---|
| HS400X callback parameter definition |

### ◆ rm_hs400x_resolutions_t

| struct rm_hs400x_resolutions_t |
|---|
| HS400X resolution block |

### ◆ rm_hs400x_raw_data_t

| struct rm_hs400x_raw_data_t | | |
|---|---|---|
| HS400X raw data | | |
| Data Fields | | |
| uint8_t | humidity[2] | Upper 2 bits of 0st element are mask. |
| uint8_t | temperature[2] | Upper 2 bits of 0st element are mask. |
| uint8_t | checksum | Checksum. |

### ◆ rm_hs400x_sensor_data_t

| struct rm_hs400x_sensor_data_t | | |
|---|---|---|
| HS400X sensor data block | | |
| Data Fields | | |
| int16_t | integer_part | |
| int16_t | decimal_part | To two decimal places. |

### ◆ rm_hs400x_data_t

| struct rm_hs400x_data_t |
|---|
| HS400X data block |

### ◆ rm_hs400x_cfg_t

| struct rm_hs400x_cfg_t | |
|---|---|
| HS400X Configuration | |
| **Data Fields** | |
| rm_hs400x_temperature_resolution_t const | temperature_resolution |
| | Resolution for temperature. |
| | |
| rm_hs400x_humidity_resolution_t const | humidity_resolution |
| | Resolution for humidity. |
| | |
| rm_hs400x_periodic_measurement_frequency_t const | frequency |
| | Frequency for periodic measurement. |
| | |
| rm_comms_instance_t const * | p_comms_instance |
| | Pointer to Communications Middleware instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void(* | p_comms_callback )(rm_hs400x_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |

### ◆ rm_hs400x_api_t

struct rm_hs400x_api_t

HS400X APIs

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | measurementStart )(rm_hs400x_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | measurementStop )(rm_hs400x_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_raw_data_t *const p_raw_data) |
| | |
| fsp_err_t(* | dataCalculate )(rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_raw_data_t *const p_raw_data, rm_hs400x_data_t *const p_hs400x_data) |
| | |
| fsp_err_t(* | close )(rm_hs400x_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_hs400x_api_t::open) (rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_cfg_t const *const p_cfg)

Open sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ measurementStart

fsp_err_t(* rm_hs400x_api_t::measurementStart) (rm_hs400x_ctrl_t *const p_ctrl)

Start one shot measurement.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ measurementStop

fsp_err_t(* rm_hs400x_api_t::measurementStop) (rm_hs400x_ctrl_t *const p_ctrl)

Stop a period measurement.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* rm_hs400x_api_t::read) (rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_raw_data_t *const p_raw_data)

Read ADC data from HS400X.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data structure. |

### ◆ dataCalculate

fsp_err_t(* rm_hs400x_api_t::dataCalculate) (rm_hs400x_ctrl_t *const p_ctrl, rm_hs400x_raw_data_t *const p_raw_data, rm_hs400x_data_t *const p_hs400x_data)

Calculate humidity and temperature values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_hs400x_data | Pointer to HS400X data structure. |

### ◆ close

fsp_err_t(* rm_hs400x_api_t::close) (rm_hs400x_ctrl_t *const p_ctrl)

Close HS400X.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_hs400x_instance_t

struct rm_hs400x_instance_t

HS400X instance

| Data Fields | | |
|---|---|---|
| rm_hs400x_ctrl_t * | p_ctrl | Pointer to the control structure for this instance |
| rm_hs400x_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance |
| rm_hs400x_api_t const * | p_api | Pointer to the API structure for this instance |

## Typedef Documentation

### ◆ rm_hs400x_ctrl_t

| typedef void rm_hs400x_ctrl_t |
|---|
| HS400X control block. Allocate an instance specific control block to pass into the HS400X API calls. |

## Enumeration Type Documentation

### ◆ rm_hs400x_event_t

| enum rm_hs400x_event_t |
|---|
| Event in the callback function |

### ◆ rm_hs400x_temperature_resolution_t

| enum rm_hs400x_temperature_resolution_t |
|---|
| Resolution type for temperature |

### ◆ rm_hs400x_humidity_resolution_t

| enum rm_hs400x_humidity_resolution_t |
|---|
| Resolution type for humidity |

### ◆ rm_hs400x_periodic_measurement_frequency_t

| enum rm_hs400x_periodic_measurement_frequency_t | |
|---|---|
| Frequency of periodic measurement | |
| Enumerator | |
| RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_2HZ | A measurement every 0.5s. |
| RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_1HZ | A measurement every 1s. |
| RM_HS400X_PERIODIC_MEASUREMENT_FREQUENCY_0P4HZ | A measurement every 2.5s. |

### 5.3.14.4 SENSOR Middleware Interface

Interfaces » Sensor

#### Detailed Description

Interface for AIR SENSOR Middleware functions.

# Summary

The AIR SENSOR interface provides AIR SENSOR functionality.

#### Data Structures

| | |
|---|---|
| struct | rm_air_sensor_callback_args_t |
| struct | rm_air_sensor_raw_data_t |
| struct | rm_air_sensor_version_t |
| struct | rm_air_sensor_single_data_t |
| struct | rm_air_sensor_data_t |
| struct | rm_air_sensor_cfg_t |
| struct | rm_air_sensor_api_t |
| struct | rm_air_sensor_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | rm_air_sensor_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_air_sensor_event_t |
| enum | rm_air_sensor_sign_t |

## Data Structure Documentation

### ◆ rm_air_sensor_callback_args_t

| struct rm_air_sensor_callback_args_t | | |
|---|---|---|
| AIR SENSOR API callback parameter definition | | |
| Data Fields | | |
| void * | p_context | |
| rm_air_sensor_event_t | event | Event that triggered the callback. |

### ◆ rm_air_sensor_raw_data_t

| struct rm_air_sensor_raw_data_t | | |
|---|---|---|
| AIR SENSOR raw data structure | | |
| Data Fields | | |
| uint8_t | results[37] | Raw sensor data array. |

### ◆ rm_air_sensor_version_t

| struct rm_air_sensor_version_t | | |
|---|---|---|
| AIR SENSOR version structure | | |
| Data Fields | | |
| uint8_t | major | Firmware major version. |
| uint8_t | minor | Firmware minor version. |
| uint8_t | patch | Firmware patch version. |
| uint8_t | ascii_code[11] | ASCII code of firmware version. |

### ◆ rm_air_sensor_single_data_t

| struct rm_air_sensor_single_data_t | | |
|---|---|---|
| AIR SENSOR sensor data block | | |
| Data Fields | | |
| rm_air_sensor_sign_t | sign | Sign of sensor data. |
| uint32_t | integer_part | Integer part of sensor data. |
| uint32_t | decimal_part | Decimal part of sensor data. |

### ◆ rm_air_sensor_data_t

| struct rm_air_sensor_data_t | | |
|---|---|---|
| AIR SENSOR data block | | |
| Data Fields | | |
| uint32_t | status | |
| rm_air_sensor_single_data_t | nc_0p3 | Number concentration of particle size 0.3 um - 10 um [1/cm3]. |
| rm_air_sensor_single_data_t | nc_0p5 | Number concentration of particle size 0.5 um - 10 um [1/cm3]. |
| rm_air_sensor_single_data_t | nc_1 | Number concentration of particle size 1 um - 10 um [1/cm3]. |
| rm_air_sensor_single_data_t | nc_2p5 | Number concentration of particle size 2.5 um - 10 um [1/cm3]. |
| rm_air_sensor_single_data_t | nc_4 | Number concentration of particle size 4 um - 10 um [1/cm3]. |
| rm_air_sensor_single_data_t | pm1_1 | Mass concentration of particle size 0.3 um - 1 um with reference to KCl particle [um/cm3]. |
| rm_air_sensor_single_data_t | pm2p5_1 | Mass concentration of particle size 0.3 um - 2.5 um with reference to KCl particle [um/cm3]. |
| rm_air_sensor_single_data_t | pm10_1 | Mass concentration of particle size 0.3 um - 10 um with reference to KCl particle [um/cm3]. |
| rm_air_sensor_single_data_t | pm1_2 | Mass concentration of particle size 0.3 um - 1 um with reference to cigarette smoke [um/cm3]. |
| rm_air_sensor_single_data_t | pm2p5_2 | Mass concentration of particle size 0.3 um - 2.5 um with reference to cigarette smoke [um/cm3]. |
| rm_air_sensor_single_data_t | pm10_2 | Mass concentration of particle size 0.3 um - 10 um with reference to cigarette smoke [um/cm3]. |

| | | |
|---|---|---|
| rm_air_sensor_single_data_t | temperature | Temperature [Celsius]. |
| rm_air_sensor_single_data_t | humidity | Humidity [RH]. |
| rm_air_sensor_single_data_t | tvoc | Total volatile organic compounds (TVOC) concentrations [mg/m3]. |
| rm_air_sensor_single_data_t | eco2 | Carbon dioxide (CO2) level [ppm]. Value is estimated/actual value depends on sensor module. |
| rm_air_sensor_single_data_t | iaq | Indoor Air Quality level according to UBA. |
| rm_air_sensor_single_data_t | rel_iaq | Relative IAQ. |

### ◆ rm_air_sensor_cfg_t

| struct rm_air_sensor_cfg_t | |
|---|---|
| AIR SENSOR configuration block | |
| **Data Fields** | |
| rm_comms_instance_t const * | p_comms_instance |
| | Pointer to Communications Middleware instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void(* | p_comms_callback )(rm_air_sensor_callback_args_t *p_args) |
| | Communications callback. |
| | |

### ◆ rm_air_sensor_api_t

| struct rm_air_sensor_api_t | |
|---|---|
| AIR SENSOR APIs | |
| **Data Fields** | |
| | |

| | |
|---|---|
| fsp_err_t(* | open )(rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | statusCheck )(rm_air_sensor_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_raw_data_t *const p_raw_data) |
| | |
| fsp_err_t(* | dataCalculate )(rm_air_sensor_ctrl_t *const p_ctrl, const rm_air_sensor_raw_data_t *const p_raw_data, rm_air_sensor_data_t *const p_sensor_data) |
| | |
| fsp_err_t(* | firmwareVersionGet )(rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_version_t *const p_version) |
| | |
| fsp_err_t(* | algorithmVersionGet )(rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_version_t *const p_version) |
| | |
| fsp_err_t(* | close )(rm_air_sensor_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_air_sensor_api_t::open) (rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_cfg_t const *const p_cfg)

Open sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ statusCheck

fsp_err_t(* rm_air_sensor_api_t::statusCheck) (rm_air_sensor_ctrl_t *const p_ctrl)

Read status of the sensor

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* rm_air_sensor_api_t::read) (rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_raw_data_t *const p_raw_data)

Read ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_raw_data | Pointer to raw data structure. |

### ◆ dataCalculate

fsp_err_t(* rm_air_sensor_api_t::dataCalculate) (rm_air_sensor_ctrl_t *const p_ctrl, const rm_air_sensor_raw_data_t *const p_raw_data, rm_air_sensor_data_t *const p_sensor_data)

Calculate values from measurement results.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [out] | p_sensor_data | Pointer to SENSOR data structure. |

### ◆ firmwareVersionGet

fsp_err_t(* rm_air_sensor_api_t::firmwareVersionGet) (rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_version_t *const p_version)

Read firmware version.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_version | Pointer to firmware version structure. |

### ◆ algorithmVersionGet

| fsp_err_t(* rm_air_sensor_api_t::algorithmVersionGet) (rm_air_sensor_ctrl_t *const p_ctrl, rm_air_sensor_version_t *const p_version) |
|---|

Read algorithm version.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_version | Pointer to algorithm version structure. |

### ◆ close

| fsp_err_t(* rm_air_sensor_api_t::close) (rm_air_sensor_ctrl_t *const p_ctrl) |
|---|

Close the sensor

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_air_sensor_instance_t

| struct rm_air_sensor_instance_t | | |
|---|---|---|
| AIR SENSOR instance | | |
| Data Fields | | |
| rm_air_sensor_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_air_sensor_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_air_sensor_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_air_sensor_ctrl_t

| typedef void rm_air_sensor_ctrl_t |
|---|
| AIR SENSOR Control block. Allocate an instance specific control block to pass into the API calls. |

## Enumeration Type Documentation

◆ **rm_air_sensor_event_t**

| enum rm_air_sensor_event_t | |
|---|---|
| Event in the callback function | |
| Enumerator | |
| RM_AIR_SENSOR_EVENT_SUCCESS | The operation completed successfully. |
| RM_AIR_SENSOR_EVENT_MEASUREMENT_COMPLETE | Sensor data is ready. |
| RM_AIR_SENSOR_EVENT_MEASUREMENT_NOT_COMPLETE | Sensor measurement is in progress. |
| RM_AIR_SENSOR_EVENT_ERROR | A communication error has occurred. |
| RM_AIR_SENSOR_EVENT_NOT_MEASURABLE | Dust concentration is extremely high or has accumulated inside the sensor. |
| RM_AIR_SENSOR_EVENT_FAN_SPEED_OUT_OF_RANGE | Fan speed is out of the set range. |
| RM_AIR_SENSOR_EVENT_FAN_BROKEN | Fan malfunctioned or broken down. |
| RM_AIR_SENSOR_EVENT_CHECKSUM_ERROR | Checksum error occurs. |

◆ **rm_air_sensor_sign_t**

| enum rm_air_sensor_sign_t | |
|---|---|
| Sign of sensor data | |
| Enumerator | |
| RM_AIR_SENSOR_SIGN_MINUS | Minus sign. |
| RM_AIR_SENSOR_SIGN_PLUS | Plus sign. |

### 5.3.14.5 ZMOD4XXX Middleware Interface

Interfaces » Sensor

#### Detailed Description

Interface for ZMOD4XXX Middleware functions.

# Summary

The ZMOD4XXX interface provides ZMOD4XXX functionality.

### Data Structures

| | |
|---|---|
| struct | rm_zmod4xxx_callback_args_t |
| struct | rm_zmod4xxx_raw_data_t |
| struct | rm_zmod4xxx_iaq_1st_data_t |
| struct | rm_zmod4xxx_iaq_2nd_data_t |
| struct | rm_zmod4xxx_sulfur_odor_data_t |
| struct | rm_zmod4xxx_oaq_2nd_data_t |
| struct | rm_zmod4xxx_raq_data_t |
| struct | rm_zmod4xxx_rel_iaq_data_t |
| struct | rm_zmod4xxx_pbaq_data_t |
| struct | rm_zmod4xxx_no2_o3_data_t |
| struct | rm_zmod4xxx_cfg_t |
| struct | rm_zmod4xxx_api_t |
| struct | rm_zmod4xxx_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_zmod4xxx_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_zmod4xxx_event_t |
| enum | rm_zmod4xxx_sulfur_odor_t |

## Data Structure Documentation

### ◆ rm_zmod4xxx_callback_args_t

| struct rm_zmod4xxx_callback_args_t | | |
|---|---|---|
| ZMOD4XXX sensor API callback parameter definition | | |
| Data Fields | | |
| void * | p_context | Pointer to data provided by the user. |

| | | |
|---|---|---|
| rm_zmod4xxx_event_t | event | Callback event. |

### ◆ rm_zmod4xxx_raw_data_t

| struct rm_zmod4xxx_raw_data_t | | |
|---|---|---|
| ZMOD4XXX raw data structure | | |
| Data Fields | | |
| uint8_t | adc_data[32] | In RRH46410, this means measurement results. |

### ◆ rm_zmod4xxx_iaq_1st_data_t

| struct rm_zmod4xxx_iaq_1st_data_t | | |
|---|---|---|
| ZMOD4XXX IAQ 1st gen data structure | | |
| Data Fields | | |
| float | rmox | MOx resistance. |
| float | rcda | CDA resistance. |
| float | iaq | IAQ index. |
| float | tvoc | TVOC concentration (mg/m^3). |
| float | etoh | EtOH concentration (ppm). |
| float | eco2 | eCO2 concentration (ppm). |

### ◆ rm_zmod4xxx_iaq_2nd_data_t

| struct rm_zmod4xxx_iaq_2nd_data_t | | |
|---|---|---|
| ZMOD4XXX IAQ 2nd gen data structure | | |
| Data Fields | | |
| float | rmox[13] | MOx resistance. |
| float | compensation_rmox | MOx resistance for compensation. |
| float | log_rcda | log10 of CDA resistance for IAQ 2nd Gen. |
| float | log_nonlog_rcda[3] | log10 of CDA resistance for IAQ 2nd Gen ULP. |
| float | iaq | IAQ index. |
| float | tvoc | TVOC concentration (mg/m^3). |
| float | etoh | EtOH concentration (ppm). |
| float | eco2 | eCO2 concentration (ppm). |
| uint8_t | sample_id | Sample ID. RRH46410 only. |
| float | rel_iaq | relative IAQ index. |

### ◆ rm_zmod4xxx_sulfur_odor_data_t

| struct rm_zmod4xxx_sulfur_odor_data_t | | |
|---|---|---|
| ZMOD4XXX Sulfur-Odor structure | | |
| Data Fields | | |
| float | rmox[9] | MOx resistance. |
| float | intensity | odor intensity rating ranges from 0.0 to 5.0 for sulfur lib. |
| rm_zmod4xxx_sulfur_odor_t | odor | sulfur_odor classification for lib. |

### ◆ rm_zmod4xxx_oaq_2nd_data_t

| struct rm_zmod4xxx_oaq_2nd_data_t | | |
|---|---|---|
| ZMOD4XXX OAQ 2nd gen data structure | | |
| Data Fields | | |
| float | rmox[8] | MOx resistance. |
| float | temperature | Temperature (degC) used for ambient compensation. |
| float | ozone_concentration | The ozone concentration in part-per-billion. |
| uint16_t | fast_aqi | 1-minute average of the Air Quality Index according to the EPA standard based on ozone. |
| uint16_t | epa_aqi | The Air Quality Index according to the EPA standard based on ozone. |

### ◆ rm_zmod4xxx_raq_data_t

| struct rm_zmod4xxx_raq_data_t | | |
|---|---|---|
| ZMOD4XXX RAQ structure | | |
| Data Fields | | |
| bool | control_signal | Control signal input for raq lib. |
| float | raq | Concentration ratio for raq lib. |

### ◆ rm_zmod4xxx_rel_iaq_data_t

| struct rm_zmod4xxx_rel_iaq_data_t | | |
|---|---|---|
| ZMOD4XXX Relative IAQ data structure | | |
| Data Fields | | |
| float | rmox[13] | MOx resistances. |
| float | rhtr | heater resistance. |
| float | rel_iaq | relative IAQ index. |

### ◆ rm_zmod4xxx_pbaq_data_t

| struct rm_zmod4xxx_pbaq_data_t | | |
|---|---|---|
| ZMOD4XXX PBAQ data structure | | |
| Data Fields | | |
| float | rmox[13] | MOx resistance. |
| float | compensation_rmox | MOx resistance for compensation. |
| float | log_rcda | log10 of CDA resistance. |
| float | rhtr | heater resistance. |
| float | temperature | ambient temperature (degC). |
| float | tvoc | TVOC concentration (mg/m^3). |
| float | etoh | EtOH concentration (ppm). |
| float | iaq | IAQ index. |
| float | rel_iaq | relative IAQ index. |
| uint8_t | sample_id | Sample ID. RRH46410 only. |

### ◆ rm_zmod4xxx_no2_o3_data_t

| struct rm_zmod4xxx_no2_o3_data_t | | |
|---|---|---|
| ZMOD4XXX NO2 O3 data structure | | |
| Data Fields | | |
| float | rmox[4] | MOx resistance. |
| float | temperature | Temperature (degC) used for ambient compensation. |
| float | ozone_concentration | O3_conc_ppb stands for the ozone concentration in part-per-billion. |
| float | no2_concentration | NO2_conc_ppb stands for the NO2 concentration in part-per-billion. |
| uint16_t | fast_aqi | FAST_AQI stands for a 1-minute average of the Air Quality Index according to the EPA standard based on ozone. |
| uint16_t | epa_aqi | EPA_AQI stands for the Air Quality Index according to the EPA standard based on ozone. |

### ◆ rm_zmod4xxx_cfg_t

| struct rm_zmod4xxx_cfg_t |
|---|
| ZMOD4XXX configuration block |

| **Data Fields** | |
|---|---|
| rm_comms_instance_t const * | p_comms_instance |
| | Pointer to Communications Middleware instance. |
| | |
| void const * | p_irq_instance |
| | Pointer to IRQ instance. |
| | |
| void * | p_context |
| | Pointer to the user-provided context. |
| | |
| void const * | p_extend |
| | Pointer to extended configuration by instance of interface. |
| | |
| void(* | p_comms_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | I2C Communications callback. |
| | |
| void(* | p_irq_callback )(rm_zmod4xxx_callback_args_t *p_args) |
| | IRQ callback. |
| | |

### ◆ rm_zmod4xxx_api_t

| struct rm_zmod4xxx_api_t | |
|---|---|
| ZMOD4XXX APIs | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | measurementStart )(rm_zmod4xxx_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | measurementStop )(rm_zmod4xxx_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | statusCheck )(rm_zmod4xxx_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data) |
| | |
| fsp_err_t(* | iaq1stGenDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | iaq2ndGenDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | sulfurOdorDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | oaq2ndGenDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | raqDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_raq_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | rellaqDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_rel_iaq_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | pbaqDataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_pbaq_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | no2O3DataCalculate )(rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_no2_o3_data_t *const p_zmod4xxx_data) |
| | |
| fsp_err_t(* | temperatureAndHumiditySet )(rm_zmod4xxx_ctrl_t *const p_ctrl, float temperature, float humidity) |
| | |

| | |
|---|---|
| fsp_err_t(* | deviceErrorCheck )(rm_zmod4xxx_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(rm_zmod4xxx_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_zmod4xxx_api_t::open) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg)

Open sensor.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ measurementStart

fsp_err_t(* rm_zmod4xxx_api_t::measurementStart) (rm_zmod4xxx_ctrl_t *const p_ctrl)

Start measurement

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ measurementStop

fsp_err_t(* rm_zmod4xxx_api_t::measurementStop) (rm_zmod4xxx_ctrl_t *const p_ctrl)

Stop measurement

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ statusCheck

fsp_err_t(* rm_zmod4xxx_api_t::statusCheck) (rm_zmod4xxx_ctrl_t *const p_ctrl)

Read status of the sensor

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ read

fsp_err_t(* rm_zmod4xxx_api_t::read) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data)

Read ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data structure. |

### ◆ iaq1stGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::iaq1stGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data)

Calculate IAQ 1st Gen. values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ iaq2ndGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::iaq2ndGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data)

Calculate IAQ 2nd Gen. values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ sulfurOdorDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::sulfurOdorDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data)

Calculate Sulfur Odor values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ oaq2ndGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::oaq2ndGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data)

Calculate OAQ 2nd Gen. values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ raqDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::raqDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_raq_data_t *const p_zmod4xxx_data)

Calculate RAQ values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ rellaqDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::rellaqDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_rel_iaq_data_t *const p_zmod4xxx_data)

Calculate Relative IAQ values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ pbaqDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::pbaqDataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_pbaq_data_t *const p_zmod4xxx_data)

Calculate PBAQ values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ no2O3DataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::no2O3DataCalculate) (rm_zmod4xxx_ctrl_t *const p_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_no2_o3_data_t *const p_zmod4xxx_data)

Calculate NO2 O3 values from ADC data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_raw_data | Pointer to raw data. |
| [in] | p_zmod4xxx_data | Pointer to ZMOD4XXXX data structure. |

### ◆ temperatureAndHumiditySet

fsp_err_t(* rm_zmod4xxx_api_t::temperatureAndHumiditySet) (rm_zmod4xxx_ctrl_t *const p_ctrl, float temperature, float humidity)

Set temperature and humidity.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | temperature | Temperature (deg C). |
| [in] | humidity | Humidity (percent). |

### ◆ deviceErrorCheck

fsp_err_t(* rm_zmod4xxx_api_t::deviceErrorCheck) (rm_zmod4xxx_ctrl_t *const p_ctrl)

Check device error event.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ close

fsp_err_t(* rm_zmod4xxx_api_t::close) (rm_zmod4xxx_ctrl_t *const p_ctrl)

Close the sensor

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_zmod4xxx_instance_t

struct rm_zmod4xxx_instance_t

ZMOD4XXX instance

Data Fields

| | | |
|---|---|---|
| rm_zmod4xxx_ctrl_t * | p_ctrl | Pointer to the control structure for this instance |
| rm_zmod4xxx_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance |
| rm_zmod4xxx_api_t const * | p_api | Pointer to the API structure for this instance |

## Typedef Documentation

### ◆ rm_zmod4xxx_ctrl_t

| typedef void rm_zmod4xxx_ctrl_t |
|---|
| ZMOD4xxx Control block. Allocate an instance specific control block to pass into the API calls. |

#### Enumeration Type Documentation

### ◆ rm_zmod4xxx_event_t

| enum rm_zmod4xxx_event_t | |
|---|---|
| Event in the callback function | |
| Enumerator | |
| RM_ZMOD4XXX_EVENT_SUCCESS | Operation was completed successfully. |
| RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE | Sensor measurement results read successfully. |
| RM_ZMOD4XXX_EVENT_MEASUREMENT_NOT_COMPLETE | Sensor measurement in progress. |
| RM_ZMOD4XXX_EVENT_DEV_ERR_POWER_ON_RESET | Unexpected reset. |
| RM_ZMOD4XXX_EVENT_DEV_ERR_ACCESS_CONFLICT | Getting invalid results while results readout. |
| RM_ZMOD4XXX_EVENT_DEV_ERR_DAMAGE | Sensor may be damaged. |
| RM_ZMOD4XXX_EVENT_ERROR | A communication error has occurred. |

### ◆ rm_zmod4xxx_sulfur_odor_t

| enum rm_zmod4xxx_sulfur_odor_t |
|---|
| Sulfur-Odor status |

## 5.3.15 Storage

Interfaces

#### Detailed Description

Storage Interfaces.

### Modules

| | |
|---|---|
| | Block Media Interface<br>Interface for block media memory access. |
| | FileX Block Media Port Interface<br>Interface for FileX Block Media port. |
| | Flash Interface<br>Interface for the Flash Memory. |
| | FreeRTOS+FAT Port Interface<br>Interface for FreeRTOS+FAT port. |
| | LittleFS Interface<br>Interface for LittleFS access. |
| | SD/MMC Interface<br>Interface for accessing SD, eMMC, and SDIO devices. |
| | SPI Flash Interface<br>Interface for accessing external SPI flash devices. |
| | Virtual EEPROM Interface<br>Interface for Virtual EEPROM access. |

#### 5.3.15.1 Block Media Interface

Interfaces » Storage

### Detailed Description

Interface for block media memory access.

# Summary

The block media interface supports reading, writing, and erasing media devices. All functions are non-blocking if possible. The callback is used to determine when an operation completes.

## Data Structures

| | |
|---|---|
| struct | rm_block_media_info_t |
| struct | rm_block_media_callback_args_t |
| struct | rm_block_media_cfg_t |
| struct | rm_block_media_status_t |
| struct | rm_block_media_api_t |
| struct | rm_block_media_instance_t |

## Typedefs

| | |
|---|---|
| typedef void | rm_block_media_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | rm_block_media_event_t |

## Data Structure Documentation

### ◆ rm_block_media_info_t

| struct rm_block_media_info_t | | |
|---|---|---|
| Block media device information supported by the instance | | |
| Data Fields | | |
| uint32_t | sector_size_bytes | Sector size in bytes. |
| uint32_t | num_sectors | Total number of sectors. |
| bool | reentrant | True if connected block media driver is reentrant. |
| bool | write_protected | True if connected block media device is write protected. |

### ◆ rm_block_media_callback_args_t

| struct rm_block_media_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rm_block_media_event_t | event | The event can be used to identify what caused the callback. |
| | | |

| | | |
|---|---|---|
| void * | p_context | Placeholder for user data. |

### ◆ rm_block_media_cfg_t

| struct rm_block_media_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| void(* | p_callback )(rm_block_media_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

### ◆ rm_block_media_status_t

| struct rm_block_media_status_t | | |
|---|---|---|
| Current status | | |
| Data Fields | | |
| bool | initialized | False if rm_block_media_api_t::mediaInit has not been called since media was inserted, true otherwise. |
| bool | busy | True if media is busy with a previous write/erase operation. |
| bool | media_inserted | Media insertion status, true if media is not removable. |

### ◆ rm_block_media_api_t

| struct rm_block_media_api_t | |
|---|---|
| Block media interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg) |
| | |

| | |
|---|---|
| fsp_err_t(* | mediaInit )(rm_block_media_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | read )(rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | write )(rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | erase )(rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks) |
| | |
| fsp_err_t(* | callbackSet )(rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | statusGet )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status) |
| | |
| fsp_err_t(* | infoGet )(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(rm_block_media_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_block_media_api_t::open) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)

Initialize block media device. rm_block_media_api_t::mediaInit must be called to complete the initialization procedure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ mediaInit

fsp_err_t(* rm_block_media_api_t::mediaInit) (rm_block_media_ctrl_t *const p_ctrl)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |

### ◆ read

fsp_err_t(* rm_block_media_api_t::read) (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)

Reads blocks of data from the specified memory device address to the location specified by the caller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_dest_address | Destination to read the data into. |
| [in] | block_address | Block address to read the data from. |
| [in] | num_blocks | Number of blocks of data to read. |

### ◆ write

fsp_err_t(* rm_block_media_api_t::write) (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)

Writes blocks of data to the specified device memory address.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [in] | p_src_address | Address to read the data to be written. |
| [in] | block_address | Block address to write the data to. |
| [in] | num_blocks | Number of blocks of data to write. |

### ◆ erase

fsp_err_t(* rm_block_media_api_t::erase) (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)

Erases blocks of data from the memory device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [in] | block_address | Block address to start the erase process at. |
| [in] | num_blocks | Number of blocks of data to erase. |

### ◆ callbackSet

fsp_err_t(* rm_block_media_api_t::callbackSet) (rm_block_media_ctrl_t *const p_ctrl, void( *p_callback)(rm_block_media_callback_args_t *), void *const p_context, rm_block_media_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ statusGet

fsp_err_t(* rm_block_media_api_t::statusGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)

Get status of connected device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_status | Pointer to store current status. |

### ◆ infoGet

| fsp_err_t(* rm_block_media_api_t::infoGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info) |
|---|

Returns information about the block media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |
| [out] | p_info | Pointer to information structure. All elements of this structure will be set by the function. |

### ◆ close

| fsp_err_t(* rm_block_media_api_t::close) (rm_block_media_ctrl_t *const p_ctrl) |
|---|

Closes the module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_block_media_api_t::open call. |

### ◆ rm_block_media_instance_t

| struct rm_block_media_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_block_media_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_block_media_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_block_media_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_block_media_ctrl_t

| typedef void rm_block_media_ctrl_t |
|---|
| Block media API control block. Allocate an instance specific control block to pass into the block media API calls. |

### Enumeration Type Documentation

#### ◆ rm_block_media_event_t

| enum rm_block_media_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED | Media removed event. |
| RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED | Media inserted event. |
| RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE | Read, write, or erase completed. |
| RM_BLOCK_MEDIA_EVENT_ERROR | Error on media operation. |
| RM_BLOCK_MEDIA_EVENT_POLL_STATUS | Poll rm_block_media_api_t::statusGet for write/erase completion. |
| RM_BLOCK_MEDIA_EVENT_MEDIA_SUSPEND | Media suspended event. |
| RM_BLOCK_MEDIA_EVENT_MEDIA_RESUME | Media resumed event. |
| RM_BLOCK_MEDIA_EVENT_WAIT | Indication to user that they should wait for an interrupt on a pending operation. |
| RM_BLOCK_MEDIA_EVENT_WAIT_END | Indication to user that interrupt has been received and waiting can end. |

## 5.3.15.2 FileX Block Media Port Interface

Interfaces » Storage

### Detailed Description

Interface for FileX Block Media port.

# Summary

The FileX block media port provides notifications for insertion and removal of removable media and provides initialization functions required by FileX.

### Data Structures

| | |
|---|---|
| struct | rm_filex_block_media_callback_args_t |

| | |
|---|---|
| struct | rm_filex_block_media_cfg_t |
| struct | rm_filex_block_media_api_t |
| struct | rm_filex_block_media_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_filex_block_media_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_filex_block_media_partition_t |

## Data Structure Documentation

### ◆ rm_filex_block_media_callback_args_t

| struct rm_filex_block_media_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rm_block_media_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. |

### ◆ rm_filex_block_media_cfg_t

| struct rm_filex_block_media_cfg_t | |
|---|---|
| Block media configuration structure | |
| **Data Fields** | |
| rm_block_media_instance_t * | p_lower_lvl_block_media |
| | Lower level block media pointer. |
| | |
| rm_filex_block_media_partition_t | partition |
| | Partition to use for partitioned media. |
| | |
| void(* | p_callback )(rm_filex_block_media_callback_args_t *p_args) |
| | Pointer to callback function. |

### ◆ rm_filex_block_media_api_t

struct rm_filex_block_media_api_t

FileX block media functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_filex_block_media_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* rm_filex_block_media_api_t::open) (rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg)

Open media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* rm_filex_block_media_api_t::close) (rm_filex_block_media_ctrl_t *const p_ctrl)

Close media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_filex_block_media_instance_t

struct rm_filex_block_media_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_filex_block_media_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_filex_block_media_cfg_t const *const | p_cfg | Pointer to the configuration structure for this instance. |

| | | |
|---|---|---|
| rm_filex_block_media_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ rm_filex_block_media_ctrl_t

| typedef void rm_filex_block_media_ctrl_t |
|---|
| Block media control structure |

### Enumeration Type Documentation

#### ◆ rm_filex_block_media_partition_t

| enum rm_filex_block_media_partition_t | |
|---|---|
| Partitions that can be selected to use FileX with | |
| Enumerator | |
| RM_FILEX_BLOCK_MEDIA_PARTITION0 | Partition 0 (0x01BE) in Master Boot Record Partition Table. |
| RM_FILEX_BLOCK_MEDIA_PARTITION1 | Partition 1 (0x01CE) in Master Boot Record Partition Table. |
| RM_FILEX_BLOCK_MEDIA_PARTITION2 | Partition 2 (0x01DE) in Master Boot Record Partition Table. |
| RM_FILEX_BLOCK_MEDIA_PARTITION3 | Partition 3 (0x01EE) in Master Boot Record Partition Table. |

## 5.3.15.3 Flash Interface

Interfaces » Storage

### Detailed Description

Interface for the Flash Memory.

# Summary

The Flash interface provides the ability to read, write, erase, and blank check the code flash and data flash regions.

### Data Structures

| | |
|---|---|
| struct | flash_block_info_t |
| struct | flash_regions_t |
| struct | flash_info_t |
| struct | flash_callback_args_t |
| struct | flash_cfg_t |
| struct | flash_api_t |
| struct | flash_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | flash_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | flash_result_t |
| enum | flash_startup_area_swap_t |
| enum | flash_event_t |
| enum | flash_id_code_mode_t |
| enum | flash_status_t |
| enum | flash_arc_t |

## Data Structure Documentation

### ◆ flash_block_info_t

struct flash_block_info_t

Flash block details stored in factory flash.

| Data Fields | | |
|---|---|---|
| uint32_t | block_section_st_addr | Starting address for this block section (blocks of this size) |
| uint32_t | block_section_end_addr | Ending address for this block section (blocks of this size) |
| uint32_t | block_size | Flash erase block size. |
| uint32_t | block_size_write | Flash write block size. |

### ◆ flash_regions_t

| struct flash_regions_t | | |
|---|---|---|
| Flash block details | | |
| Data Fields | | |
| uint32_t | num_regions | Length of block info array. |
| flash_block_info_t const * | p_block_array | Block info array base address. |

### ◆ flash_info_t

| struct flash_info_t | | |
|---|---|---|
| Information about the flash blocks | | |
| Data Fields | | |
| flash_regions_t | code_flash | Information about the code flash regions. |
| flash_regions_t | data_flash | Information about the code flash regions. |

### ◆ flash_callback_args_t

| struct flash_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| flash_event_t | event | Event can be used to identify what caused the callback (flash ready or error). |
| uint32_t | data | Event specific data. Invalid unless specified by the event. |
| void * | p_context | Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t. |

### ◆ flash_cfg_t

| struct flash_cfg_t | |
|---|---|
| FLASH Configuration | |
| **Data Fields** | |
| bool | data_flash_bgo |
| | True if BGO (Background Operation) is enabled for Data Flash. |
| | |
| void(* | p_callback )(flash_callback_args_t *p_args) |
| | Callback provided when a Flash interrupt ISR occurs. |
| | |

| | |
|---|---|
| void const * | p_extend |
| | FLASH hardware dependent configuration. |
| | |
| void * | p_context |
| | Placeholder for user data. Passed to user callback in flash_callback_args_t. |
| | |
| uint8_t | ipl |
| | Flash ready interrupt priority. |
| | |
| IRQn_Type | irq |
| | Flash ready interrupt number. |
| | |
| uint8_t | err_ipl |
| | Flash error interrupt priority (unused in r_flash_lp) |
| | |
| IRQn_Type | err_irq |
| | Flash error interrupt number (unused in r_flash_lp) |
| | |

◆ **flash_api_t**

| struct flash_api_t | |
|---|---|
| Shared Interface definition for FLASH | |
| **Data Fields** | |
| fsp_err_t(* | open )(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | write )(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes) |
| | |
| fsp_err_t(* | erase )(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks) |
| | |

| | |
|---|---|
| fsp_err_t(* | blankCheck )(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result) |
| | |
| fsp_err_t(* | infoGet )(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | statusGet )(flash_ctrl_t *const p_ctrl, flash_status_t *const p_status) |
| | |
| fsp_err_t(* | accessWindowSet )(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr) |
| | |
| fsp_err_t(* | accessWindowClear )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | idCodeSet )(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode) |
| | |
| fsp_err_t(* | reset )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | updateFlashClockFreq )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | startupAreaSelect )(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary) |
| | |
| fsp_err_t(* | bankSwap )(flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(flash_ctrl_t *const p_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | antiRollbackCounterIncrement )(flash_ctrl_t *const p_ctrl, flash_arc_t counter) |
| | |
| fsp_err_t(* | antiRollbackCounterRefresh )(flash_ctrl_t *const p_ctrl, flash_arc_t counter) |
| | |

| | |
|---|---|
| fsp_err_t(* | antiRollbackCounterRead )(flash_ctrl_t *const p_ctrl, flash_arc_t counter, uint32_t *const p_count) |
| | |
| fsp_err_t(* | userLockableAreaWrite )(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

Open FLASH device.

**Parameters**

| | | |
|---|---|---|
| [out] | p_ctrl | Pointer to FLASH device control. Must be declared by user. Value set here. |
| [in] | flash_cfg_t | Pointer to FLASH configuration structure. All elements of this structure must be set by the user. |

### ◆ write

fsp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device context. |
| [in] | src_address | Address of the buffer containing the data to write to Flash. |
| [in] | flash_address | Code Flash or Data Flash address to write. The address must be on a programming line boundary. |
| [in] | num_bytes | The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF. |

**Warning**

Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_BYTES being returned and no data written.

### ◆ erase

fsp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Erase FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device. |
| [in] | address | The block containing this address is the first block erased. |
| [in] | num_blocks | Specifies the number of blocks to be erased, the starting block determined by the block_erase_address. |

### ◆ blankCheck

fsp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)

Blank check FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control for the FLASH device context. |
| [in] | address | The starting address of the Flash area to blank check. |
| [in] | num_bytes | Specifies the number of bytes that need to be checked. See the specific handler for details. |
| [out] | p_blank_check_result | Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result. |

### ◆ infoGet

fsp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

Close FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [out] | p_info | Pointer to FLASH info structure. |

### ◆ close

fsp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)

Close FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ statusGet

fsp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)

Get Status for FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [out] | p_ctrl | Pointer to the current flash status. |

### ◆ accessWindowSet

fsp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

Set Access Window for FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | start_addr | Determines the Starting block for the Code Flash access window. |
| [in] | end_addr | Determines the Ending block for the Code Flash access window. This address will not be within the access window. |

### ◆ accessWindowClear

fsp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)

Clear any existing Code Flash access window for FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | start_addr | Determines the Starting block for the Code Flash access window. |
| [in] | end_addr | Determines the Ending block for the Code Flash access window. |

### ◆ idCodeSet

fsp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode.

For example, uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0

With mode FLASH_ID_CODE_MODE_LOCKED, it will result in an ID code of 00112233445566778899aabbccddee80

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | p_id_bytes | Ponter to the ID Code to be written. |
| [in] | mode | Mode used for checking the ID code. |

### ◆ reset

fsp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)

Reset function for FLASH device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ updateFlashClockFreq

fsp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)

Update Flash clock frequency (FCLK) and recalculate timeout values

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ startupAreaSelect

fsp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | swap_type | FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG. |
| [in] | is_temporary | True or false. See table below. |

| swap_type | is_temporary | Operation |
|---|---|---|
| FLASH_STARTUP_AREA_BLOCK0 | false | On next reset Startup area will be Block 0. |
| FLASH_STARTUP_AREA_BLOCK1 | true | Startup area is immediately, but temporarily switched to Block 1. |
| FLASH_STARTUP_AREA_BTFLG | true | Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG. |

### ◆ bankSwap

fsp_err_t(* flash_api_t::bankSwap) (flash_ctrl_t *const p_ctrl)

Swap the bank used as the startup area. On Flash HP, need to change into dual bank mode to use this feature.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |

### ◆ callbackSet

fsp_err_t(* flash_api_t::callbackSet) (flash_ctrl_t *const p_ctrl, void(*p_callback)(flash_callback_args_t *), void *const p_context, flash_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ antiRollbackCounterIncrement

fsp_err_t(* flash_api_t::antiRollbackCounterIncrement) (flash_ctrl_t *const p_ctrl, flash_arc_t counter)

Increment the selected anti-rollback counter.

**Warning**

Once incremented, the counter can never be decremented or reset, even with a debug probe.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | counter | The anti-rollback counter to increment |

### ◆ antiRollbackCounterRefresh

fsp_err_t(* flash_api_t::antiRollbackCounterRefresh) (flash_ctrl_t *const p_ctrl, flash_arc_t counter)

Refresh the selected anti-rollback counter flash area.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | counter | The anti-rollback counter to refresh |

### ◆ antiRollbackCounterRead

fsp_err_t(* flash_api_t::antiRollbackCounterRead) (flash_ctrl_t *const p_ctrl, flash_arc_t counter, uint32_t *const p_count)

Read current count value of the selected anti-rollback counter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | counter | The anti-rollback counter to read |
| [out] | p_count | The current count value |

### ◆ userLockableAreaWrite

fsp_err_t(* flash_api_t::userLockableAreaWrite) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write to the user lockable area for the flash device

**Warning**

Data will not be written if any segment of the specified area is locked.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to FLASH device control. |
| [in] | src_address | Address of the buffer containing the data to write to flash. |
| [in] | flash_address | User lockable area flash address to write. The address must be on a programming line boundary. |
| [in] | num_bytes | The number of bytes to write. This number must be a multiple of the programming size. |

### ◆ flash_instance_t

struct flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| flash_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| flash_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| flash_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ flash_ctrl_t

typedef void flash_ctrl_t

Flash control block. Allocate an instance specific control block to pass into the flash API calls.

## Enumeration Type Documentation

### ◆ flash_result_t

| enum flash_result_t | |
|---|---|
| Result type for certain operations | |
| Enumerator | |
| FLASH_RESULT_BLANK | Return status for Blank Check Function. |
| FLASH_RESULT_NOT_BLANK | Return status for Blank Check Function. |
| FLASH_RESULT_BGO_ACTIVE | Flash is configured for BGO mode. Result is returned in callback. |

### ◆ flash_startup_area_swap_t

| enum flash_startup_area_swap_t | |
|---|---|
| Parameter for specifying the startup area swap being requested by startupAreaSelect() | |
| Enumerator | |
| FLASH_STARTUP_AREA_BTFLG | Startup area will be set based on the value of the BTFLG. |
| FLASH_STARTUP_AREA_BLOCK0 | Startup area will be set to Block 0. |
| FLASH_STARTUP_AREA_BLOCK1 | Startup area will be set to Block 1. |

### ◆ flash_event_t

| enum flash_event_t | |
|---|---|
| Event types returned by the ISR callback when used in Data Flash BGO mode | |
| Enumerator | |
| FLASH_EVENT_ERASE_COMPLETE | Erase operation successfully completed. |
| FLASH_EVENT_WRITE_COMPLETE | Write operation successfully completed. |
| FLASH_EVENT_BLANK | Blank check operation successfully completed. Specified area is blank. |
| FLASH_EVENT_NOT_BLANK | Blank check operation successfully completed. Specified area is NOT blank. |
| FLASH_EVENT_ERR_DF_ACCESS | Data Flash operation failed. Can occur when writing an unerased section. |
| FLASH_EVENT_ERR_CF_ACCESS | Code Flash operation failed. Can occur when writing an unerased section. |
| FLASH_EVENT_ERR_CMD_LOCKED | Operation failed, FCU is in Locked state (often result of an illegal command) |
| FLASH_EVENT_ERR_FAILURE | Erase or Program Operation failed. |
| FLASH_EVENT_ERR_ONE_BIT | A 1-bit error has been corrected when reading the flash memory area by the sequencer. See callback 'data' field for address. |
| FLASH_EVENT_ERR_TWO_BIT | A 2-bit error has been corrected when reading the flash memory area by the sequencer. See callback 'data' field for address. |
| FLASH_EVENT_ERR_ECC | An uncorrectable error has been detected while reading memory. See callback 'data' field for address. |

### ◆ flash_id_code_mode_t

| enum flash_id_code_mode_t | |
|---|---|
| ID Code Modes for writing to ID code registers | |
| Enumerator | |
| FLASH_ID_CODE_MODE_UNLOCKED | ID code is ignored. |
| FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT | ID code is checked. All erase is available. |
| FLASH_ID_CODE_MODE_LOCKED | ID code is checked. |

### ◆ flash_status_t

| enum flash_status_t | |
|---|---|
| Flash status | |
| Enumerator | |
| FLASH_STATUS_IDLE | The flash is idle. |
| FLASH_STATUS_BUSY | The flash is currently processing a command. |

### ◆ flash_arc_t

| enum flash_arc_t | |
|---|---|
| Anti-rollback counter selection. | |
| Enumerator | |
| FLASH_ARC_SEC | Anti-Rollback Counter for the secure application. |
| FLASH_ARC_OEMBL | Anti-Rollback Counter for the OEM bootloader. |
| FLASH_ARC_NSEC_0 | Anti-Rollback Counter for non-secure application 0 (use when a single non-secure counter is available or when multiple non-secure counters are available) |
| FLASH_ARC_NSEC_1 | Anti-Rollback Counter for non-secure application 1 (use when multiple non-secure counters are available) |
| FLASH_ARC_NSEC_2 | Anti-Rollback Counter for non-secure application 2 (use when multiple non-secure counters are available) |
| FLASH_ARC_NSEC_3 | Anti-Rollback Counter for non-secure application 3 (use when multiple non-secure counters are available) |

## 5.3.15.4 FreeRTOS+FAT Port Interface

Interfaces » Storage

### Detailed Description

Interface for FreeRTOS+FAT port.

# Summary

The FreeRTOS+FAT port provides notifications for insertion and removal of removable media and provides initialization functions required by FreeRTOS+FAT.

### Data Structures

| | |
|---|---|
| struct | rm_freertos_plus_fat_callback_args_t |
| struct | rm_freertos_plus_fat_device_t |

| | |
|---|---|
| struct | rm_freertos_plus_fat_info_t |
| struct | rm_freertos_plus_fat_cfg_t |
| struct | rm_freertos_plus_fat_disk_cfg_t |
| struct | rm_freertos_plus_fat_api_t |
| struct | rm_freertos_plus_fat_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_freertos_plus_fat_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_freertos_plus_fat_event_t |
| enum | rm_freertos_plus_fat_type_t |

## Data Structure Documentation

### ◆ rm_freertos_plus_fat_callback_args_t

| struct rm_freertos_plus_fat_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rm_freertos_plus_fat_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. |

### ◆ rm_freertos_plus_fat_device_t

| struct rm_freertos_plus_fat_device_t | | |
|---|---|---|
| Information obtained from the media device. | | |
| Data Fields | | |
| uint32_t | sector_count | Sector count. |
| uint32_t | sector_size_bytes | Sector size in bytes. |

### ◆ rm_freertos_plus_fat_info_t

| struct rm_freertos_plus_fat_info_t | | |
|---|---|---|
| Basic information about a FAT volume. | | |
| Data Fields | | |
| uint32_t | sector_size | Sector size. |

| | | |
|---|---|---|
| uint32_t | sectors_per_cluster | Sectors per cluster. |
| uint32_t | total_sectors | Total number of sectors. |
| uint32_t | free_sectors | Current free sectors. |
| uint8_t * | p_volume_label | Volume label. |
| rm_freertos_plus_fat_type_t | type | FAT type for this disk. |
| uint8_t | partition_number | Partition number for this disk. |

### ◆ rm_freertos_plus_fat_cfg_t

| struct rm_freertos_plus_fat_cfg_t | |
|---|---|
| FreeRTOS+FAT configuration structure. | |
| **Data Fields** | |
| void(* | p_callback )(rm_freertos_plus_fat_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| fsp_err_t(* | p_busy_callback )(void *p_busy_context) |
| | |
| void * | p_busy_context |
| | User defined context passed into busy callback function. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

## Field Documentation

### ◆ p_busy_callback

| fsp_err_t(* rm_freertos_plus_fat_cfg_t::p_busy_callback) (void *p_busy_context) |
|---|
| The busy callback function is called when the device is busy with a write or erase operation. It can be used to pend the current thread while waiting for the operation to complete and to manage timeouts if required. Set to NULL if pending the thread is not required and the thread will block waiting for the device busy status to clear. If this function returns an error (anything except FSP_SUCCESS), the upper layer function returns FF_ERR_IOMAN_DRIVER_FATAL_ERROR. |

### ◆ rm_freertos_plus_fat_disk_cfg_t

| struct rm_freertos_plus_fat_disk_cfg_t | | |
|---|---|---|
| Initializing a FAT disk. | | |
| Data Fields | | |
| rm_freertos_plus_fat_device_t | device | Device sector data. |
| uint32_t | cache_size_bytes | Chache memory size. |
| uint8_t * | p_cache | Pointer to cache memory. |
| uint8_t | partition_number | Partition number for this disk. |

### ◆ rm_freertos_plus_fat_api_t

| struct rm_freertos_plus_fat_api_t | |
|---|---|
| FreeRTOS+FAT functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device) |
| | |
| fsp_err_t(* | diskInit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk) |
| | |
| fsp_err_t(* | diskDeinit )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk) |
| | |
| fsp_err_t(* | infoGet )(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info) |
| | |
| fsp_err_t(* | close )(rm_freertos_plus_fat_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_freertos_plus_fat_api_t::open) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)

Open media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ mediaInit

fsp_err_t(* rm_freertos_plus_fat_api_t::mediaInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_device | Pointer to store device information. |

### ◆ diskInit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)

Initializes a FreeRTOS+FAT FF_Disk_t structure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_disk_cfg | Pointer to disk configurations |
| [out] | p_disk | Pointer to store FreeRTOS+FAT disk structure. |

### ◆ diskDeinit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskDeinit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)

Deinitializes a FreeRTOS+FAT FF_Disk_t structure.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_disk_cfg | Pointer to disk configurations |
| [out] | p_disk | Pointer to store FreeRTOS+FAT disk structure. |

### ◆ infoGet

fsp_err_t(* rm_freertos_plus_fat_api_t::infoGet) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)

Returns information about the media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [out] | p_info | Pointer to information structure. All elements of this structure will be set by the function. |

### ◆ close

fsp_err_t(* rm_freertos_plus_fat_api_t::close) (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Close media device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ rm_freertos_plus_fat_instance_t

struct rm_freertos_plus_fat_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rm_freertos_plus_fat_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_freertos_plus_fat_cfg_t const *const | p_cfg | Pointer to the configuration structure for this instance. |

| | | |
|---|---|---|
| rm_freertos_plus_fat_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ rm_freertos_plus_fat_ctrl_t

| typedef void rm_freertos_plus_fat_ctrl_t |
|---|
| FreeRTOS+FAT control structure |

### Enumeration Type Documentation

#### ◆ rm_freertos_plus_fat_event_t

| enum rm_freertos_plus_fat_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED | Media removed event. |
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED | Media inserted event. |
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_SUSPEND | Media suspended event. |
| RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_RESUME | Media resumed event. |

#### ◆ rm_freertos_plus_fat_type_t

| enum rm_freertos_plus_fat_type_t | |
|---|---|
| FAT filesystem types. | |
| Enumerator | |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT32 | FAT32 disk. |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT16 | FAT16 disk. |
| RM_FREERTOS_PLUS_FAT_TYPE_FAT12 | FAT12 disk. |

## 5.3.15.5 LittleFS Interface

Interfaces » Storage

### Detailed Description

Interface for LittleFS access.

# Summary

The LittleFS Port configures a fail-safe filesystem designed for microcontrollers on top of a lower level storage device.

### Data Structures

| | |
|---|---|
| struct | rm_littlefs_cfg_t |
| struct | rm_littlefs_api_t |
| struct | rm_littlefs_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_littlefs_ctrl_t |

### Data Structure Documentation

#### ◆ rm_littlefs_cfg_t

| struct rm_littlefs_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| struct lfs_config const * | p_lfs_cfg | Pointer LittleFS configuration structure. |
| void const * | p_extend | Pointer to hardware dependent configuration. |

#### ◆ rm_littlefs_api_t

| struct rm_littlefs_api_t | |
|---|---|
| LittleFS Port interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rm_littlefs_ctrl_t *const p_ctrl) |
| | |
| **Field Documentation** | |

### ◆ open

| fsp_err_t(* rm_littlefs_api_t::open) (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg) |
|---|

Initialize The lower level storage device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ close

| fsp_err_t(* rm_littlefs_api_t::close) (rm_littlefs_ctrl_t *const p_ctrl) |
|---|

Closes the module and lower level storage device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_littlefs_api_t::open call. |

### ◆ rm_littlefs_instance_t

| struct rm_littlefs_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| rm_littlefs_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_littlefs_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_littlefs_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_littlefs_ctrl_t

| typedef void rm_littlefs_ctrl_t |
|---|

LittleFS Port API control block. Allocate an instance specific control block to pass into the LittleFS Port API calls.

### 5.3.15.6 SD/MMC Interface

Interfaces » Storage

#### Detailed Description

Interface for accessing SD, eMMC, and SDIO devices.

## Summary

The r_sdhi interface provides standard SD and eMMC media functionality. This interface also supports SDIO.

#### Data Structures

| | |
|---|---|
| struct | sdmmc_response_t.status_b |
| struct | sdmmc_status_t |
| struct | sdmmc_device_t |
| struct | sdmmc_callback_args_t |
| struct | sdmmc_write_io_args_t |
| struct | sdmmc_read_io_ext_args_t |
| struct | sdmmc_write_io_ext_args_t |
| struct | sdmmc_cfg_t |
| struct | sdmmc_api_t |
| struct | sdmmc_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | sdmmc_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | sdmmc_card_type_t |
| enum | sdmmc_bus_width_t |
| enum | sdmmc_io_transfer_mode_t |
| enum | sdmmc_io_address_mode_t |

| | |
|---|---|
| enum | sdmmc_io_write_mode_t |
| enum | sdmmc_event_t |
| enum | sdmmc_card_detect_t |
| enum | sdmmc_write_protect_t |
| enum | sdmmc_r1_state_t |

## Data Structure Documentation

### ◆ sdmmc_response_t.status_b

| struct sdmmc_response_t.status_b |
|---|
| SDIO Card Status Register. |

### ◆ sdmmc_status_t

| struct sdmmc_status_t | | |
|---|---|---|
| Current status. | | |
| Data Fields | | |
| bool | initialized | False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected), true otherwise.<br><br>If ready is false, call sdmmc_api_t::mediaInit to reinitialize it |
| bool | transfer_in_progress | true = Card is busy |
| bool | card_inserted | Card detect status, true if card detect is not used. |

### ◆ sdmmc_device_t

| struct sdmmc_device_t | | |
|---|---|---|
| Information obtained from the media device. | | |
| Data Fields | | |
| sdmmc_card_type_t | card_type | SD, eMMC, or SDIO. |
| bool | write_protected | true = Card is write protected |
| uint32_t | clock_rate | Current clock rate. |
| uint32_t | sector_count | Sector count. |
| uint32_t | sector_size_bytes | Sector size. |

| | | |
|---|---|---|
| uint32_t | erase_sector_count | Minimum erasable unit (in 512 byte sectors) |

### ◆ sdmmc_callback_args_t

| struct sdmmc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| sdmmc_event_t | event | The event can be used to identify what caused the callback. |
| sdmmc_response_t | response | Response from card, only valid if SDMMC_EVENT_RESPONSE is set in event. |
| void * | p_context | Placeholder for user data. |

### ◆ sdmmc_write_io_args_t

| struct sdmmc_write_io_args_t |
|---|
| Non-secure arguments for writeIo guard function |

### ◆ sdmmc_read_io_ext_args_t

| struct sdmmc_read_io_ext_args_t |
|---|
| Non-secure arguments for readIoExt guard function |

### ◆ sdmmc_write_io_ext_args_t

| struct sdmmc_write_io_ext_args_t |
|---|
| Non-secure arguments for writeIoExt guard function |

### ◆ sdmmc_cfg_t

| struct sdmmc_cfg_t | |
|---|---|
| SD/MMC Configuration | |
| **Data Fields** | |
| uint8_t | channel |
| | Channel of SD/MMC host interface. |
| | |
| sdmmc_bus_width_t | bus_width |
| | Device bus width is 1, 4 or 8 bits wide. |

| | |
|---|---|
| | |
| transfer_instance_t const * | p_lower_lvl_transfer |
| | Transfer instance used to move data with DMA or DTC. |
| | |
| void(* | p_callback )(sdmmc_callback_args_t *p_args) |
| | Pointer to callback function. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |
| | SD/MMC hardware dependent configuration. |
| | |
| uint32_t | block_size |
| | |
| sdmmc_card_detect_t | card_detect |
| | |
| sdmmc_write_protect_t | write_protect |
| | |
| IRQn_Type | access_irq |
| | Access IRQ number. |
| | |
| IRQn_Type | sdio_irq |
| | SDIO IRQ number. |
| | |
| IRQn_Type | card_irq |
| | Card IRQ number. |
| | |
| IRQn_Type | dma_req_irq |

| | |
|---|---|
| | DMA request IRQ number. |
| | |
| uint8_t | access_ipl |
| | Access interrupt priority. |
| | |
| uint8_t | sdio_ipl |
| | SDIO interrupt priority. |
| | |
| uint8_t | card_ipl |
| | Card interrupt priority. |
| | |
| uint8_t | dma_req_ipl |
| | DMA request interrupt priority. |
| | |

## Field Documentation

### ◆ block_size

uint32_t sdmmc_cfg_t::block_size

Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

### ◆ card_detect

sdmmc_card_detect_t sdmmc_cfg_t::card_detect

Whether or not card detection is used.

### ◆ write_protect

sdmmc_write_protect_t sdmmc_cfg_t::write_protect

Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

### ◆ sdmmc_api_t

struct sdmmc_api_t

SD/MMC functions implemented at the HAL layer API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | mediaInit )(sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device) |
| | |
| fsp_err_t(* | read )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count) |
| | |
| fsp_err_t(* | write )(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count) |
| | |
| fsp_err_t(* | readIo )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address) |
| | |
| fsp_err_t(* | writeIo )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write) |
| | |
| fsp_err_t(* | readIoExt )(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode) |
| | |
| fsp_err_t(* | writeIoExt )(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode) |
| | |
| fsp_err_t(* | ioIntEnable )(sdmmc_ctrl_t *const p_ctrl, bool enable) |
| | |
| fsp_err_t(* | statusGet )(sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status) |
| | |
| fsp_err_t(* | erase )(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count) |
| | |
| fsp_err_t(* | callbackSet )(sdmmc_ctrl_t *const p_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory) |
| | |

| fsp_err_t(* | close )(sdmmc_ctrl_t *const p_ctrl) |
|---|---|

## Field Documentation

### ◆ open

fsp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

Open the SD/MMC driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to SD/MMC instance control block. |
| [in] | p_cfg | Pointer to SD/MMC instance configuration structure. |

### ◆ mediaInit

fsp_err_t(* sdmmc_api_t::mediaInit) (sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)

Initializes an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to SD/MMC instance control block. |
| [out] | p_device | Pointer to store device information. |

### ◆ read

fsp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_dest | Pointer to data buffer to read data to. |
| [in] | start_sector | First sector address to read. |
| [in] | sector_count | Number of sectors to read. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ write

fsp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)

Write data to SD/MMC channel. This API is not supported for SDIO devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | p_source | Pointer to data buffer to write data from. |
| [in] | start_sector | First sector address to write to. |
| [in] | sector_count | Number of sectors to write. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ readIo

fsp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_data | Pointer to location to store data byte. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |

### ◆ writeIo

fsp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in,out] | p_data | Pointer to data byte to write. Read data is also provided here if read_after_write is true. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | read_after_write | Whether or not to read back the same register after writing |

### ◆ readIoExt

fsp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_dest | Pointer to data buffer to read data to. |
| [in] | function | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | count | Number of bytes or blocks to read, maximum 512 bytes or 511 blocks. |
| [in] | transfer_mode | Byte or block mode |
| [in] | address_mode | Fixed or incrementing address mode |

### ◆ writeIoExt

fsp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | p_source | Pointer to data buffer to write data from. |
| [in] | function_number | SDIO Function Number. |
| [in] | address | SDIO register address. |
| [in] | count | Number of bytes or blocks to write, maximum 512 bytes or 511 blocks. |
| [in] | transfer_mode | Byte or block mode |
| [in] | address_mode | Fixed or incrementing address mode |

### ◆ ioIntEnable

fsp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | enable | Interrupt enable = true, interrupt disable = false. |

### ◆ statusGet

fsp_err_t(* sdmmc_api_t::statusGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)

Get SD/MMC device status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [out] | p_status | Pointer to current driver status. |

### ◆ erase

fsp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |
| [in] | start_sector | First sector to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. |
| [in] | sector_count | Number of sectors to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. All sectors must be in the range of sdmmc_device_t::sector_count. |

### ◆ callbackSet

fsp_err_t(* sdmmc_api_t::callbackSet) (sdmmc_ctrl_t *const p_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void *const p_context, sdmmc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in sdmmc_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)

Close open SD/MMC device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to an open SD/MMC instance control block. |

### ◆ sdmmc_instance_t

struct sdmmc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| sdmmc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| sdmmc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| sdmmc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ sdmmc_ctrl_t

| typedef void sdmmc_ctrl_t |
| --- |
| SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. |

## Enumeration Type Documentation

### ◆ sdmmc_card_type_t

| enum sdmmc_card_type_t | |
| --- | --- |
| SD/MMC media uses SD protocol or MMC protocol. | |
| Enumerator | |
| SDMMC_CARD_TYPE_MMC | The media is an eMMC device. |
| SDMMC_CARD_TYPE_SD | The media is an SD card. |
| SDMMC_CARD_TYPE_SDIO | The media is an SDIO card. |

### ◆ sdmmc_bus_width_t

| enum sdmmc_bus_width_t | |
| --- | --- |
| SD/MMC data bus is 1, 4 or 8 bits wide. | |
| Enumerator | |
| SDMMC_BUS_WIDTH_1_BIT | Data bus is 1 bit wide. |
| SDMMC_BUS_WIDTH_4_BITS | Data bus is 4 bits wide. |
| SDMMC_BUS_WIDTH_8_BITS | Data bus is 8 bits wide. |

### ◆ sdmmc_io_transfer_mode_t

| enum sdmmc_io_transfer_mode_t | |
| --- | --- |
| SDIO transfer mode, configurable in SDIO read/write extended commands. | |
| Enumerator | |
| SDMMC_IO_MODE_TRANSFER_BYTE | SDIO byte transfer mode. |
| SDMMC_IO_MODE_TRANSFER_BLOCK | SDIO block transfer mode. |

### ◆ sdmmc_io_address_mode_t

| enum sdmmc_io_address_mode_t | |
|---|---|
| SDIO address mode, configurable in SDIO read/write extended commands. | |
| Enumerator | |
| SDMMC_IO_ADDRESS_MODE_FIXED | Write all data to the same address. |
| SDMMC_IO_ADDRESS_MODE_INCREMENT | Increment destination address after each write. |

### ◆ sdmmc_io_write_mode_t

| enum sdmmc_io_write_mode_t | |
|---|---|
| Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register. | |
| Enumerator | |
| SDMMC_IO_WRITE_MODE_NO_READ | Write only (do not read back) |
| SDMMC_IO_WRITE_READ_AFTER_WRITE | Read back the register after write. |

### ◆ sdmmc_event_t

| enum sdmmc_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| SDMMC_EVENT_CARD_REMOVED | Card removed event. |
| SDMMC_EVENT_CARD_INSERTED | Card inserted event. |
| SDMMC_EVENT_RESPONSE | Response event. |
| SDMMC_EVENT_SDIO | IO event. |
| SDMMC_EVENT_TRANSFER_COMPLETE | Read or write complete. |
| SDMMC_EVENT_TRANSFER_ERROR | Read or write failed. |
| SDMMC_EVENT_ERASE_COMPLETE | Erase completed. |
| SDMMC_EVENT_ERASE_BUSY | Erase timeout, poll sdmmc_api_t::statusGet. |

### ◆ sdmmc_card_detect_t

| enum sdmmc_card_detect_t | |
|---|---|
| Card detection configuration options. | |
| Enumerator | |
| SDMMC_CARD_DETECT_NONE | Card detection unused. |
| SDMMC_CARD_DETECT_CD | Card detection using the CD pin. |

### ◆ sdmmc_write_protect_t

| enum sdmmc_write_protect_t | |
|---|---|
| Write protection configuration options. | |
| Enumerator | |
| SDMMC_WRITE_PROTECT_NONE | Write protection unused. |
| SDMMC_WRITE_PROTECT_WP | Write protection using WP pin. |

### ◆ sdmmc_r1_state_t

| enum sdmmc_r1_state_t | |
|---|---|
| Card state when receiving the prior command. | |
| Enumerator | |
| SDMMC_R1_STATE_IDLE | Idle State. |
| SDMMC_R1_STATE_READY | Ready State. |
| SDMMC_R1_STATE_IDENT | Identification State. |
| SDMMC_R1_STATE_STBY | Stand-by State. |
| SDMMC_R1_STATE_TRAN | Transfer State. |
| SDMMC_R1_STATE_DATA | Sending-data State. |
| SDMMC_R1_STATE_RCV | Receive-data State. |
| SDMMC_R1_STATE_PRG | Programming State. |
| SDMMC_R1_STATE_DIS | Disconnect State (between programming and stand-by) |
| SDMMC_R1_STATE_IO | This is an I/O card and memory states do not apply. |

## 5.3.15.7 SPI Flash Interface

Interfaces » Storage

### Detailed Description

Interface for accessing external SPI flash devices.

# Summary

The SPI flash API provides an interface that configures, writes, and erases sectors in SPI flash devices.

### Data Structures

| | |
|---|---|
| struct | spi_flash_erase_command_t |

| | |
|---|---|
| struct | spi_flash_direct_transfer_t |
| struct | spi_flash_cfg_t |
| struct | spi_flash_status_t |
| struct | spi_flash_api_t |
| struct | spi_flash_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | spi_flash_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | spi_flash_read_mode_t |
| enum | spi_flash_protocol_t |
| enum | spi_flash_address_bytes_t |
| enum | spi_flash_data_lines_t |
| enum | spi_flash_dummy_clocks_t |
| enum | spi_flash_direct_transfer_dir_t |

## Data Structure Documentation

### ◆ spi_flash_erase_command_t

| struct spi_flash_erase_command_t | | |
|---|---|---|
| Structure to define an erase command and associated erase size. | | |
| Data Fields | | |
| uint16_t | command | Erase command. |
| uint32_t | size | Size of erase for associated command, set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE for chip erase. |

### ◆ spi_flash_direct_transfer_t

| struct spi_flash_direct_transfer_t | | |
|---|---|---|
| Structure to define a direct transfer. | | |
| Data Fields | | |
| union spi_flash_direct_transfer_t | __unnamed__ | |

| | | |
|---|---|---|
| uint32_t | address | Starting address. |
| uint16_t | command | Transfer command. |
| uint8_t | dummy_cycles | Number of dummy cycles. |
| uint8_t | command_length | Command length. |
| uint8_t | address_length | Address length. |
| uint8_t | data_length | Data length. |

### ◆ spi_flash_cfg_t

| struct spi_flash_cfg_t | | |
|---|---|---|
| User configuration structure used by the open function | | |
| Data Fields | | |
| spi_flash_protocol_t | spi_protocol | Initial SPI protocol. SPI protocol can be changed in spi_flash_api_t::spiProtocolSet. |
| spi_flash_read_mode_t | read_mode | Read mode. |
| spi_flash_address_bytes_t | address_bytes | Number of bytes used to represent the address. |
| spi_flash_dummy_clocks_t | dummy_clocks | Number of dummy clocks to use for fast read operations. |
| spi_flash_data_lines_t | page_program_address_lines | Number of lines used to send address for page program command. This should either be 1 or match the number of lines used in the selected read mode. |
| uint8_t | write_status_bit | Which bit determines write status. |
| uint8_t | write_enable_bit | Which bit determines write status. |
| uint32_t | page_size_bytes | Page size in bytes (maximum number of bytes for page program). Used to specify single continuous write size (bytes) in case of OSPI RAM. |
| uint8_t | page_program_command | Page program command. |
| uint8_t | write_enable_command | Command to enable write or erase, typically 0x06. |
| uint8_t | status_command | Command to read the write status. |
| uint8_t | read_command | Read command - OSPI SPI mode only. |
| | | |

| | | |
|---|---|---|
| uint8_t | xip_enter_command | Command to enter XIP mode. |
| uint8_t | xip_exit_command | Command to exit XIP mode. |
| uint8_t | erase_command_list_length | Length of erase command list. |
| spi_flash_erase_command_t const * | p_erase_command_list | List of all erase commands and associated sizes. |
| void const * | p_extend | Pointer to implementation specific extended configurations. |

### ◆ spi_flash_status_t

| struct spi_flash_status_t | | |
|---|---|---|
| Status. | | |
| Data Fields | | |
| bool | write_in_progress | Whether or not a write is in progress. This is determined by reading the spi_flash_cfg_t::write_status_bit from the spi_flash_cfg_t::status_command. |

### ◆ spi_flash_api_t

| struct spi_flash_api_t | |
|---|---|
| SPI flash implementations follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | directWrite )(spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write) |
| | |
| fsp_err_t(* | directRead )(spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes) |
| | |
| fsp_err_t(* | directTransfer )(spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction) |
| | |
| fsp_err_t(* | spiProtocolSet )(spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol) |
| | |

| | |
|---|---|
| fsp_err_t(* | write )(spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count) |
| | |
| fsp_err_t(* | erase )(spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count) |
| | |
| fsp_err_t(* | statusGet )(spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status) |
| | |
| fsp_err_t(* | xipEnter )(spi_flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | xipExit )(spi_flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | bankSet )(spi_flash_ctrl_t *const p_ctrl, uint32_t bank) |
| | |
| fsp_err_t(* | autoCalibrate )(spi_flash_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(spi_flash_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* spi_flash_api_t::open) (spi_flash_ctrl_t *const p_ctrl, spi_flash_cfg_t const *const p_cfg)

Open the SPI flash driver module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_cfg | Pointer to a configuration structure |

### ◆ directWrite

fsp_err_t(* spi_flash_api_t::directWrite) (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)

Write raw data to the SPI flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_src | Pointer to raw data to write, must include any required command/address |
| [in] | bytes | Number of bytes to write |
| [in] | read_after_write | If true, the slave select remains asserted and the peripheral does not return to direct communications mode. If false, the slave select is deasserted and memory mapped access is possible after this function returns if the device is not busy. |

### ◆ directRead

fsp_err_t(* spi_flash_api_t::directRead) (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read raw data from the SPI flash. Must follow a call to spi_flash_api_t::directWrite.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [out] | p_dest | Pointer to read raw data into |
| [in] | bytes | Number of bytes to read |

### ◆ directTransfer

fsp_err_t(* spi_flash_api_t::directTransfer) (spi_flash_ctrl_t *const p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)

Direct Read/Write raw data to the SPI flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_data | Pointer to command, address and data values and lengths |
| [in] | direction | Direct Read/Write |

### ◆ spiProtocolSet

fsp_err_t(* spi_flash_api_t::spiProtocolSet) (spi_flash_ctrl_t *const p_ctrl, spi_flash_protocol_t spi_protocol)

Change the SPI protocol in the driver. The application must change the SPI protocol on the device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | spi_protocol | Desired SPI protocol |

### ◆ write

fsp_err_t(* spi_flash_api_t::write) (spi_flash_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)

Program a page of data to the flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_src | The memory address of the data to write to the flash device |
| [in] | p_dest | The location in the flash device address space to write the data to |
| [in] | byte_count | The number of bytes to write |

### ◆ erase

fsp_err_t(* spi_flash_api_t::erase) (spi_flash_ctrl_t *const p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)

Erase a certain number of bytes of the flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | p_device_address | The location in the flash device address space to start the erase from |
| [in] | byte_count | The number of bytes to erase. Set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE to erase entire chip. |

### ◆ statusGet

fsp_err_t(* spi_flash_api_t::statusGet) (spi_flash_ctrl_t *const p_ctrl, spi_flash_status_t *const p_status)

Get the write or erase status of the flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [out] | p_status | Current status of the SPI flash device stored here. |

### ◆ xipEnter

fsp_err_t(* spi_flash_api_t::xipEnter) (spi_flash_ctrl_t *const p_ctrl)

Enter XIP mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ xipExit

fsp_err_t(* spi_flash_api_t::xipExit) (spi_flash_ctrl_t *const p_ctrl)

Exit XIP mode.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ bankSet

fsp_err_t(* spi_flash_api_t::bankSet) (spi_flash_ctrl_t *const p_ctrl, uint32_t bank)

Select the bank to access. See implementation for details.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |
| [in] | bank | The bank number |

### ◆ autoCalibrate

fsp_err_t(* spi_flash_api_t::autoCalibrate) (spi_flash_ctrl_t *const p_ctrl)

AutoCalibrate the SPI flash driver module. Expected to be used when auto-calibrating OSPI RAM device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ close

fsp_err_t(* spi_flash_api_t::close) (spi_flash_ctrl_t *const p_ctrl)

Close the SPI flash driver module.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to a driver handle |

### ◆ spi_flash_instance_t

struct spi_flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| spi_flash_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| spi_flash_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| spi_flash_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ spi_flash_ctrl_t

| typedef void spi_flash_ctrl_t |
| --- |
| SPI flash control block. Allocate an instance specific control block to pass into the SPI flash API calls. |

## Enumeration Type Documentation

### ◆ spi_flash_read_mode_t

| enum spi_flash_read_mode_t | |
| --- | --- |
| Read mode. | |
| Enumerator | |
| SPI_FLASH_READ_MODE_STANDARD | Standard Read Mode (no dummy cycles) |
| SPI_FLASH_READ_MODE_FAST_READ | Fast Read Mode (dummy cycles between address and data) |
| SPI_FLASH_READ_MODE_FAST_READ_DUAL_OUTPUT | Fast Read Dual Output Mode (data on 2 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_DUAL_IO | Fast Read Dual I/O Mode (address and data on 2 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_QUAD_OUTPUT | Fast Read Quad Output Mode (data on 4 lines) |
| SPI_FLASH_READ_MODE_FAST_READ_QUAD_IO | Fast Read Quad I/O Mode (address and data on 4 lines) |

### ◆ spi_flash_protocol_t

| enum spi_flash_protocol_t | |
|---|---|
| SPI protocol. | |
| Enumerator | |
| SPI_FLASH_PROTOCOL_EXTENDED_SPI | Extended SPI mode (commands on 1 line) |
| SPI_FLASH_PROTOCOL_QPI | QPI mode (commands on 4 lines). Note that the application must ensure the device is in QPI mode. |
| SPI_FLASH_PROTOCOL_SOPI | SOPI mode (command and data on 8 lines). Note that the application must ensure the device is in SOPI mode. |
| SPI_FLASH_PROTOCOL_DOPI | DOPI mode (command and data on 8 lines, dual data rate). Note that the application must ensure the device is in DOPI mode. |
| SPI_FLASH_PROTOCOL_1S_1S_1S | 1S-1S-1S protocol mode |
| SPI_FLASH_PROTOCOL_4S_4D_4D | 4S-4D-4D protocol mode |
| SPI_FLASH_PROTOCOL_8D_8D_8D | 8D-8D-8D protocol mode |
| SPI_FLASH_PROTOCOL_1S_2S_2S | 1S-2S-2S protocol mode |
| SPI_FLASH_PROTOCOL_2S_2S_2S | 2S-2S-2S protocol mode |
| SPI_FLASH_PROTOCOL_1S_4S_4S | 1S-4S-4S protocol mode |
| SPI_FLASH_PROTOCOL_4S_4S_4S | 4S-4S-4S protocol mode |

### ◆ spi_flash_address_bytes_t

| enum spi_flash_address_bytes_t | |
|---|---|
| Number of bytes in the address. | |
| Enumerator | |
| SPI_FLASH_ADDRESS_BYTES_1 | 1-byte address phase |
| SPI_FLASH_ADDRESS_BYTES_2 | 2-byte address phase |
| SPI_FLASH_ADDRESS_BYTES_3 | 3-byte address phase |
| SPI_FLASH_ADDRESS_BYTES_4 | 4-byte address phase |
| SPI_FLASH_ADDRESS_BYTES_4_4BYTE_READ_CODE | 4 address bytes using standard 4-byte command set. |

### ◆ spi_flash_data_lines_t

| enum spi_flash_data_lines_t | |
|---|---|
| Number of data lines used. | |
| Enumerator | |
| SPI_FLASH_DATA_LINES_1 | 1 data line |
| SPI_FLASH_DATA_LINES_2 | 2 data lines |
| SPI_FLASH_DATA_LINES_4 | 4 data lines |

### ◆ spi_flash_dummy_clocks_t

| enum spi_flash_dummy_clocks_t | |
|---|---|
| Number of dummy cycles for fast read operations. | |
| Enumerator | |
| SPI_FLASH_DUMMY_CLOCKS_0 | 0 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_1 | 1 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_2 | 2 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_3 | 3 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_4 | 4 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_5 | 5 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_6 | 6 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_7 | 7 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_8 | 8 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_9 | 9 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_10 | 10 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_11 | 11 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_12 | 12 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_13 | 13 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_14 | 14 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_15 | 15 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_16 | 16 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_17 | 17 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_18 | 18 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_19 | 19 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_20 | 20 dummy clocks |

| | |
|---|---|
| SPI_FLASH_DUMMY_CLOCKS_21 | 21 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_22 | 22 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_23 | 23 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_24 | 24 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_25 | 25 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_26 | 26 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_27 | 27 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_28 | 28 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_29 | 29 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_30 | 30 dummy clocks |
| SPI_FLASH_DUMMY_CLOCKS_31 | 31 dummy clocks |

### ◆ spi_flash_direct_transfer_dir_t

| |
|---|
| enum spi_flash_direct_transfer_dir_t |
| Direct Read and Write direction |

### 5.3.15.8 Virtual EEPROM Interface

Interfaces » Storage

#### Detailed Description

Interface for Virtual EEPROM access.

## Summary

The Virtual EEPROM Port configures a fail-safe key value store designed for microcontrollers on top of a lower level storage device.

#### Data Structures

| | |
|---|---|
| struct | rm_vee_callback_args_t |

| | |
|---|---|
| struct | rm_vee_cfg_t |
| struct | rm_vee_api_t |
| struct | rm_vee_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | rm_vee_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rm_vee_state_t |

## Data Structure Documentation

### ◆ rm_vee_callback_args_t

| struct rm_vee_callback_args_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| rm_vee_state_t | state | State of the Virtual EEPROM. |
| void * | p_context | Placeholder for user data. Set in rm_vee_api_t::open function in::rm_vee_cfg_t. |

### ◆ rm_vee_cfg_t

| struct rm_vee_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| uint32_t | start_addr |
| | Start address to be used for Virtual EEPROM memory. |
| | |
| uint32_t | num_segments |
| | Number of segments to divide the volume into. |
| | |
| uint32_t | total_size |
| | Total size of the volume. |
| | |
| uint32_t | ref_data_size |

| | |
|---|---|
| | Size of the reference data stored at the end of the segment. |
| | |
| uint32_t | record_max_id |
| | Maximum record ID that can be used. |
| | |
| uint16_t * | rec_offset |
| | Pointer to buffer used for record offset caching. |
| | |
| void(* | p_callback )(rm_vee_callback_args_t *p_args) |
| | Callback provided when a Virtual EEPROM event occurs. |
| | |
| void * | p_context |
| | Placeholder for user data. |
| | |
| void const * | p_extend |
| | Pointer to hardware dependent configuration. |
| | |

### ◆ rm_vee_api_t

| struct rm_vee_api_t | |
|---|---|
| Virtual EEPROM interface API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rm_vee_ctrl_t *const p_ctrl, rm_vee_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | recordWrite )(rm_vee_ctrl_t *const p_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t num_bytes) |
| | |
| fsp_err_t(* | recordPtrGet )(rm_vee_ctrl_t *const p_ctrl, uint32_t rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes) |
| | |
| fsp_err_t(* | refDataWrite )(rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data) |

| | |
|---|---|
| | |
| fsp_err_t(* | refDataPtrGet )(rm_vee_ctrl_t *const p_ctrl, uint8_t **const pp_ref_data) |
| | |
| fsp_err_t(* | statusGet )(rm_vee_ctrl_t *const p_ctrl, rm_vee_status_t *const p_status) |
| | |
| fsp_err_t(* | refresh )(rm_vee_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | format )(rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data) |
| | |
| fsp_err_t(* | callbackSet )(rm_vee_ctrl_t *const p_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void *const p_context, rm_vee_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(rm_vee_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rm_vee_api_t::open) (rm_vee_ctrl_t *const p_ctrl, rm_vee_cfg_t const *const p_cfg)

Initializes the driver's internal structures and opens the Flash driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ recordWrite

fsp_err_t(* rm_vee_api_t::recordWrite) (rm_vee_ctrl_t *const p_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t num_bytes)

Writes a record to data flash.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | rec_id | ID of record to write. |
| [in] | p_rec_data | Pointer to record data to write. |
| [in] | num_bytes | Length of data to write. |

### ◆ recordPtrGet

fsp_err_t(* rm_vee_api_t::recordPtrGet) (rm_vee_ctrl_t *const p_ctrl, uint32_t rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes)

This function gets the pointer to the most recent version of a record specified by ID.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | rec_id | ID of record to locate. |
| [in] | pp_rec_data | Pointer to set to the most recent version of the record. |
| [in] | p_num_bytes | Variable to load with record length. |

### ◆ refDataWrite

fsp_err_t(* rm_vee_api_t::refDataWrite) (rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)

Writes new Reference data to the reference update area.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | p_ref_data | Pointer to data to write to the reference data update area. |

### ◆ refDataPtrGet

fsp_err_t(* rm_vee_api_t::refDataPtrGet) (rm_vee_ctrl_t *const p_ctrl, uint8_t **const pp_ref_data)

Gets a pointer to the most recent reference data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | pp_ref_data | Pointer to set to the most recent valid reference data. |

### ◆ statusGet

fsp_err_t(* rm_vee_api_t::statusGet) (rm_vee_ctrl_t *const p_ctrl, rm_vee_status_t *const p_status)

Get the current status of the VEE driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | p_status | Pointer to store the current status of the VEE driver. |

### ◆ refresh

fsp_err_t(* rm_vee_api_t::refresh) (rm_vee_ctrl_t *const p_ctrl)

Manually start a refresh operation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |

### ◆ format

fsp_err_t(* rm_vee_api_t::format) (rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)

Format the Virtual EEPROM.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. |
| [in] | p_ref_data | Optional pointer to reference data to write during format. |

### ◆ callbackSet

fsp_err_t(* rm_vee_api_t::callbackSet) (rm_vee_ctrl_t *const p_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void *const p_context, rm_vee_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_vee_api_t::open call. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* rm_vee_api_t::close) (rm_vee_ctrl_t *const p_ctrl)

Closes the module and lower level storage device.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in rm_vee_api_t::open call. |

### ◆ rm_vee_instance_t

struct rm_vee_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| rm_vee_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rm_vee_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| rm_vee_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rm_vee_ctrl_t

| typedef void rm_vee_ctrl_t |
| --- |
| Virtual EEPROM API control block. Allocate an instance specific control block to pass into the VEE API calls. |

### Enumeration Type Documentation

### ◆ rm_vee_state_t

| enum rm_vee_state_t | |
| --- | --- |
| Enumerator | |
| RM_VEE_STATE_READY | Ready. |
| RM_VEE_STATE_BUSY | Operation in progress. |
| RM_VEE_STATE_REFRESH | Refresh operation in progress. |
| RM_VEE_STATE_OVERFLOW | The amount of data written exceeds the space available. |
| RM_VEE_STATE_HARDWARE_FAIL | Lower level hardware failure. |

## 5.3.16 System

Interfaces

### Detailed Description

System Interfaces.

### Modules

| | |
| --- | --- |
| | CGC Interface<br>Interface for clock generation. |
| | ELC Interface<br>Interface for the Event Link Controller. |
| | I/O Port Interface |

| | |
|---|---|
| | Interface for accessing I/O ports and configuring I/O functionality. |
| | IPC Interface<br>Interface for inter-processor communications (IPC). |

### 5.3.16.1 CGC Interface

Interfaces » System

#### Detailed Description

Interface for clock generation.

## Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

#### Data Structures

| | |
|---|---|
| struct | cgc_callback_args_t |
| struct | cgc_pll_cfg_t |
| struct | cgc_divider_cfg_t |
| struct | cgc_cfg_t |
| struct | cgc_clocks_cfg_t |
| struct | cgc_api_t |
| struct | cgc_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | cgc_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | cgc_event_t |
| enum | cgc_clock_t |
| enum | cgc_pll_div_t |
| enum | cgc_pll_out_div_t |
| enum | cgc_sys_clock_div_t |
| enum | cgc_pin_output_control_t |
| enum | cgc_usb_clock_div_t |
| enum | cgc_clock_change_t |

## Data Structure Documentation

### ◆ cgc_callback_args_t

| struct cgc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| cgc_event_t | event | The event can be used to identify what caused the callback. |
| void * | p_context | Placeholder for user data. |

### ◆ cgc_pll_cfg_t

| struct cgc_pll_cfg_t | | |
|---|---|---|
| Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock. | | |
| Data Fields | | |
| cgc_clock_t | source_clock | PLL source clock (main oscillator or HOCO) |
| cgc_pll_div_t | divider | PLL divider. |
| cgc_pll_mul_t | multiplier | PLL multiplier. |
| cgc_pll_out_div_t | out_div_p | PLL divisor for output clock P. |
| cgc_pll_out_div_t | out_div_q | PLL divisor for output clock Q. |
| cgc_pll_out_div_t | out_div_r | PLL divisor for output clock R. |

### ◆ cgc_divider_cfg_t

| struct cgc_divider_cfg_t |
|---|

Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions.

| Data Fields | | |
|---|---|---|
| cgc_sys_clock_div_t | moco_divider | MOCO divider. |
| cgc_sys_clock_div_t | hoco_divider | HOCO divider. |
| cgc_sys_clock_div_t | mosc_divider | Main oscillator divider. |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |
| union cgc_divider_cfg_t | __unnamed__ | |

### ◆ cgc_cfg_t

struct cgc_cfg_t

Configuration options.

**Data Fields**

| | |
|---|---|
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |

### ◆ cgc_clocks_cfg_t

struct cgc_clocks_cfg_t

Clock configuration

| Data Fields | | |
|---|---|---|
| cgc_clock_t | system_clock | System clock source enumeration. |
| cgc_pll_cfg_t | pll_cfg | PLL configuration structure. |
| cgc_pll_cfg_t | pll2_cfg | PLL2 configuration structure. |
| cgc_divider_cfg_t | divider_cfg | Clock dividers structure. |
| cgc_clock_change_t | loco_state | State of LOCO. |

| | | |
|---|---|---|
| cgc_clock_change_t | moco_state | State of MOCO. |
| cgc_clock_change_t | hoco_state | State of HOCO. |
| cgc_clock_change_t | mainosc_state | State of Main oscillator. |
| cgc_clock_change_t | pll_state | State of PLL. |
| cgc_clock_change_t | pll2_state | State of PLL2. |
| cgc_clock_change_t | subosc_state | State of Sub oscillator. |

### ◆ cgc_api_t

| struct cgc_api_t | |
|---|---|
| CGC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | clocksCfg )(cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg) |
| | |
| fsp_err_t(* | clockStart )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg) |
| | |
| fsp_err_t(* | clockStop )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| | |
| fsp_err_t(* | clockCheck )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source) |
| | |
| fsp_err_t(* | systemClockSet )(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg) |
| | |
| fsp_err_t(* | systemClockGet )(cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg) |
| | |
| fsp_err_t(* | oscStopDetectEnable )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | oscStopDetectDisable )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | oscStopStatusClear )(cgc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | callbackSet )(cgc_ctrl_t *const p_ctrl, |

| | |
|---|---|
| | void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(cgc_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* cgc_api_t::open) (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)

Initial configuration

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_cfg | Pointer to configuration |

### ◆ clocksCfg

fsp_err_t(* cgc_api_t::clocksCfg) (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)

Configure all system clocks.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | p_clock_cfg | Pointer to desired configuration of system clocks |

### ◆ clockStart

fsp_err_t(* cgc_api_t::clockStart) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)

Start a clock.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Clock source to start |
| [in] | p_pll_cfg | Pointer to PLL configuration, can be NULL if clock_source is not CGC_CLOCK_PLL or CGC_CLOCK_PLL2 |

### ◆ clockStop

fsp_err_t(* cgc_api_t::clockStop) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Stop a clock.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | The clock source to stop |

### ◆ clockCheck

fsp_err_t(* cgc_api_t::clockCheck) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Check the stability of the selected clock.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Which clock source to check for stability |

### ◆ systemClockSet

fsp_err_t(* cgc_api_t::systemClockSet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)

Set the system clock.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [in] | clock_source | Clock source to set as system clock |
| [in] | p_divider_cfg | Pointer to the clock divider configuration |

### ◆ systemClockGet

fsp_err_t(* cgc_api_t::systemClockGet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)

Get the system clock information.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |
| [out] | p_clock_source | Returns the current system clock |
| [out] | p_divider_cfg | Returns the current system clock dividers |

### ◆ oscStopDetectEnable

fsp_err_t(* cgc_api_t::oscStopDetectEnable) (cgc_ctrl_t *const p_ctrl)

Enable and optionally register a callback for Main Oscillator stop detection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ oscStopDetectDisable

fsp_err_t(* cgc_api_t::oscStopDetectDisable) (cgc_ctrl_t *const p_ctrl)

Disable Main Oscillator stop detection.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ oscStopStatusClear

fsp_err_t(* cgc_api_t::oscStopStatusClear) (cgc_ctrl_t *const p_ctrl)

Clear the oscillator stop detection flag.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to instance control block |

### ◆ callbackSet

fsp_err_t(* cgc_api_t::callbackSet) (cgc_ctrl_t *const p_ctrl, void(*p_callback)(cgc_callback_args_t *), void *const p_context, cgc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the CGC control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

| fsp_err_t(* cgc_api_t::close) (cgc_ctrl_t *const p_ctrl) |
|---|
| Close the CGC driver.<br><br>**Parameters**<br>[in] p_ctrl — Pointer to instance control block |

### ◆ cgc_instance_t

| struct cgc_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| cgc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| cgc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| cgc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ cgc_ctrl_t

| typedef void cgc_ctrl_t |
|---|
| CGC control block. Allocate an instance specific control block to pass into the CGC API calls. |

## Enumeration Type Documentation

### ◆ cgc_event_t

| enum cgc_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| CGC_EVENT_OSC_STOP_DETECT_NMI | Main oscillator stop detection has caused the NMI event. |
| CGC_EVENT_OSC_STOP_DETECT_MAIN_OSC | Main oscillator stop detection has caused the interrupt event. |
| CGC_EVENT_OSC_STOP_DETECT_SUBCLOCK | Subclock oscillator stop detection has caused the interrupt event. |

### ◆ cgc_clock_t

| enum cgc_clock_t | |
|---|---|
| System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider | |
| Enumerator | |
| CGC_CLOCK_HOCO | The high speed on chip oscillator. |
| CGC_CLOCK_MOCO | The middle speed on chip oscillator. |
| CGC_CLOCK_LOCO | The low speed on chip oscillator. |
| CGC_CLOCK_MAIN_OSC | The main oscillator. |
| CGC_CLOCK_SUBCLOCK | The subclock oscillator. |
| CGC_CLOCK_PLL | The PLL oscillator. |
| CGC_CLOCK_PLL2 | The PLL2 oscillator. |

### ◆ cgc_pll_div_t

| enum cgc_pll_div_t | |
|---|---|
| PLL divider values | |
| Enumerator | |
| CGC_PLL_DIV_1 | PLL divider of 1. |
| CGC_PLL_DIV_2 | PLL divider of 2. |
| CGC_PLL_DIV_3 | PLL divider of 3. |
| CGC_PLL_DIV_4 | PLL divider of 4. |
| CGC_PLL_DIV_6 | PLL divider of 6. |

### ◆ cgc_pll_out_div_t

| enum cgc_pll_out_div_t | |
|---|---|
| PLL clock output divisor. | |
| Enumerator | |
| CGC_PLL_OUT_DIV_2 | PLL output clock divided by 2. |
| CGC_PLL_OUT_DIV_3 | PLL output clock divided by 3. |
| CGC_PLL_OUT_DIV_4 | PLL output clock divided by 4. |
| CGC_PLL_OUT_DIV_5 | PLL output clock divided by 5. |
| CGC_PLL_OUT_DIV_6 | PLL output clock divided by 6. |
| CGC_PLL_OUT_DIV_8 | PLL output clock divided by 8. |
| CGC_PLL_OUT_DIV_9 | PLL output clock divided by 9. |
| CGC_PLL_OUT_DIV_1_5 | PLL output clock divided by 1.5. |
| CGC_PLL_OUT_DIV_16 | PLL output clock divided by 16. |

### ◆ cgc_sys_clock_div_t

| enum cgc_sys_clock_div_t | |
|---|---|
| System clock divider values - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D. | |
| Enumerator | |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| | |

| | |
|---|---|
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| | |

| | |
|---|---|
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_24 | System clock divided by 24. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| | |

| | |
|---|---|
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| | |

| | |
|---|---|
| CGC_SYS_CLOCK_DIV_24 | System clock divided by 24. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_24 | System clock divided by 24. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| | |

| | |
|---|---|
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3. |
| CGC_SYS_CLOCK_DIV_6 | System clock divided by 6. |
| CGC_SYS_CLOCK_DIV_12 | System clock divided by 12. |
| CGC_SYS_CLOCK_DIV_24 | System clock divided by 24. |
| CGC_SYS_CLOCK_DIV_1 | System clock divided by 1. |
| CGC_SYS_CLOCK_DIV_2 | System clock divided by 2. |
| CGC_SYS_CLOCK_DIV_4 | System clock divided by 4. |
| CGC_SYS_CLOCK_DIV_8 | System clock divided by 8. |
| CGC_SYS_CLOCK_DIV_16 | System clock divided by 16. |
| CGC_SYS_CLOCK_DIV_32 | System clock divided by 32. |
| CGC_SYS_CLOCK_DIV_64 | System clock divided by 64. |
| CGC_SYS_CLOCK_DIV_3 | System clock divided by 3 (BCLK only) |

### ◆ cgc_pin_output_control_t

| enum cgc_pin_output_control_t | |
|---|---|
| Pin output control enable/disable (SDCLK, BCLK). | |
| Enumerator | |
| CGC_PIN_OUTPUT_CONTROL_ENABLE | Enable pin output. |
| CGC_PIN_OUTPUT_CONTROL_DISABLE | Disable pin output. |

### ◆ cgc_usb_clock_div_t

| enum cgc_usb_clock_div_t | |
|---|---|
| USB clock divider values | |
| Enumerator | |
| CGC_USB_CLOCK_DIV_2 | Divide USB source clock by 2. |
| CGC_USB_CLOCK_DIV_3 | Divide USB source clock by 3. |
| CGC_USB_CLOCK_DIV_4 | Divide USB source clock by 4. |
| CGC_USB_CLOCK_DIV_5 | Divide USB source clock by 5. |

### ◆ cgc_clock_change_t

| enum cgc_clock_change_t | |
|---|---|
| Clock options | |
| Enumerator | |
| CGC_CLOCK_CHANGE_START | Start the clock. |
| CGC_CLOCK_CHANGE_STOP | Stop the clock. |
| CGC_CLOCK_CHANGE_NONE | No change to the clock. |

## 5.3.16.2 ELC Interface

Interfaces » System

### Detailed Description

Interface for the Event Link Controller.

### Data Structures

| | |
|---|---|
| struct | elc_cfg_t |
| struct | elc_api_t |
| struct | elc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | elc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | elc_peripheral_t |
| enum | elc_software_event_t |

## Data Structure Documentation

### ◆ elc_cfg_t

| struct elc_cfg_t | | |
|---|---|---|
| Main configuration structure for the Event Link Controller | | |
| Data Fields | | |
| elc_event_t const | link[ELC_PERIPHERAL_NUM] | Event link register settings. |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ elc_api_t

| struct elc_api_t | |
|---|---|
| ELC driver structure. General ELC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(elc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | softwareEventGenerate )(elc_ctrl_t *const p_ctrl, elc_software_event_t event_num) |
| | |
| fsp_err_t(* | linkSet )(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal) |
| | |
| fsp_err_t(* | linkBreak )(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral) |
| | |
| fsp_err_t(* | enable )(elc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(elc_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* elc_api_t::open) (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

Initialize all links in the Event Link Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to configuration structure. |

### ◆ close

fsp_err_t(* elc_api_t::close) (elc_ctrl_t *const p_ctrl)

Disable all links in the Event Link Controller and close the API.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ softwareEventGenerate

fsp_err_t(* elc_api_t::softwareEventGenerate) (elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)

Generate a software event in the Event Link Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | eventNum | Software event number to be generated. |

### ◆ linkSet

fsp_err_t(* elc_api_t::linkSet) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

Create a single event link.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | peripheral | The peripheral block that will receive the event signal. |
| [in] | signal | The event signal. |

### ◆ linkBreak

fsp_err_t(* elc_api_t::linkBreak) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

Break an event link.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | peripheral | The peripheral that should no longer be linked. |

### ◆ enable

fsp_err_t(* elc_api_t::enable) (elc_ctrl_t *const p_ctrl)

Enable the operation of the Event Link Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ disable

fsp_err_t(* elc_api_t::disable) (elc_ctrl_t *const p_ctrl)

Disable the operation of the Event Link Controller.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ elc_instance_t

struct elc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| elc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| elc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| elc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ elc_ctrl_t

| typedef void elc_ctrl_t |
|---|
| ELC control block. Allocate an instance specific control block to pass into the ELC API calls. |

#### Enumeration Type Documentation

### ◆ elc_peripheral_t

| enum elc_peripheral_t |
|---|
| Possible peripherals to be linked to event signals (not all available on all MCUs) |

### ◆ elc_software_event_t

| enum elc_software_event_t | |
|---|---|
| Software event number | |
| Enumerator | |
| ELC_SOFTWARE_EVENT_0 | Software event 0. |
| ELC_SOFTWARE_EVENT_1 | Software event 1. |

## 5.3.16.3 I/O Port Interface

Interfaces » System

#### Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

# Summary

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

#### Data Structures

| | |
|---|---|
| struct | ioport_pin_cfg_t |
| struct | ioport_cfg_t |
| struct | ioport_api_t |

| | |
|---|---|
| struct | ioport_instance_t |

### Typedefs

| | |
|---|---|
| typedef uint16_t | ioport_size_t<br>IO port size. More... |
| typedef void | ioport_ctrl_t |

## Data Structure Documentation

### ◆ ioport_pin_cfg_t

| struct ioport_pin_cfg_t | | |
|---|---|---|
| Pin identifier and pin configuration value | | |
| Data Fields | | |
| uint32_t | pin_cfg | Pin configuration - Use ioport_cfg_options_t parameters to configure. |
| bsp_io_port_pin_t | pin | Pin identifier. |

### ◆ ioport_cfg_t

| struct ioport_cfg_t | | |
|---|---|---|
| Multiple pin configuration data for loading into registers by R_IOPORT_Open() | | |
| Data Fields | | |
| uint16_t | number_of_pins | Number of pins for which there is configuration data. |
| ioport_pin_cfg_t const * | p_pin_cfg_data | Pin configuration data. |
| const void * | p_extend | Pointer to hardware extend configuration. |

### ◆ ioport_api_t

| struct ioport_api_t | |
|---|---|
| IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |
| | |
| fsp_err_t(* | close )(ioport_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | pinsCfg )(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg) |

| | |
|---|---|
| fsp_err_t(* | pinCfg )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg) |
| fsp_err_t(* | pinEventInputRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event) |
| fsp_err_t(* | pinEventOutputWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value) |
| fsp_err_t(* | pinRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value) |
| fsp_err_t(* | pinWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level) |
| fsp_err_t(* | portDirectionSet )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask) |
| fsp_err_t(* | portEventInputRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data) |
| fsp_err_t(* | portEventOutputWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value) |
| fsp_err_t(* | portRead )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value) |
| fsp_err_t(* | portWrite )(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask) |

## Field Documentation

### ◆ open

fsp_err_t(* ioport_api_t::open) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use ioport_api_t::pinsCfg for runtime reconfiguration of multiple pins.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to pin configuration data array. |

### ◆ close

fsp_err_t(* ioport_api_t::close) (ioport_ctrl_t *const p_ctrl)

Close the API.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |

### ◆ pinsCfg

fsp_err_t(* ioport_api_t::pinsCfg) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Configure multiple pins.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | p_cfg | Pointer to pin configuration data array. |

### ◆ pinCfg

fsp_err_t(* ioport_api_t::pinCfg) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)

Configure settings for an individual pin.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pin | Pin to be read. |
| [in] | cfg | Configuration options for the pin. |

### ◆ pinEventInputRead

fsp_err_t(* ioport_api_t::pinEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)

Read the event input data of the specified pin and return the level.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pin | Pin to be read. |
| [in] | p_pin_event | Pointer to return the event data. |

### ◆ pinEventOutputWrite

fsp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)

Write pin event data.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pin | Pin event data is to be written to. |
| [in] | pin_value | Level to be written to pin output event. |

### ◆ pinRead

fsp_err_t(* ioport_api_t::pinRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)

Read level of a pin.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pin | Pin to be read. |
| [in] | p_pin_value | Pointer to return the pin level. |

### ◆ pinWrite

fsp_err_t(* ioport_api_t::pinWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)

Write specified level to a pin.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | pin | Pin to be written to. |
| [in] | level | State to be written to the pin. |

### ◆ portDirectionSet

fsp_err_t(* ioport_api_t::portDirectionSet) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)

Set the direction of one or more pins on a port.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | port | Port being configured. |
| [in] | direction_values | Value controlling direction of pins on port. |
| [in] | mask | Mask controlling which pins on the port are to be configured. |

### ◆ portEventInputRead

fsp_err_t(* ioport_api_t::portEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)

Read captured event data for a port.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | port | Port to be read. |
| [in] | p_event_data | Pointer to return the event data. |

### ◆ portEventOutputWrite

fsp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

Write event output data for a port.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | port | Port event data will be written to. |
| [in] | event_data | Data to be written as event data to specified port. |
| [in] | mask_value | Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port. |

### ◆ portRead

fsp_err_t(* ioport_api_t::portRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)

Read states of pins on the specified port.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | port | Port to be read. |
| [in] | p_port_value | Pointer to return the port value. |

### ◆ portWrite

fsp_err_t(* ioport_api_t::portWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Write to multiple pins on a port.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control structure. |
| [in] | port | Port to be written to. |
| [in] | value | Value to be written to the port. |
| [in] | mask | Mask controlling which pins on the port are written to. |

### ◆ ioport_instance_t

| struct ioport_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| ioport_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ioport_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ioport_api_t const * | p_api | Pointer to the API structure for this instance. |

#### Typedef Documentation

### ◆ ioport_size_t

| typedef uint16_t ioport_size_t |
|---|
| IO port size.<br>IO port type used with ports |

### ◆ ioport_ctrl_t

| typedef void ioport_ctrl_t |
|---|
| IOPORT control block. Allocate an instance specific control block to pass into the IOPORT API calls. |

## 5.3.16.4 IPC Interface

Interfaces » System

#### Detailed Description

Interface for inter-processor communications (IPC).

# Summary

The IPC interface provides common APIs for IPC HAL drivers. The IPC interface supports the following features:

- Sending and receiving messages between cores via IPC one-way FIFOs
- Triggering maskable interrupts between cores

#### Data Structures

| | |
|---|---|
| struct | ipc_callback_args_t |
| struct | ipc_cfg_t |
| struct | ipc_api_t |
| struct | ipc_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | ipc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | ipc_event_t |
| enum | ipc_generate_event_t |

## Data Structure Documentation

### ◆ ipc_callback_args_t

| struct ipc_callback_args_t | | |
|---|---|---|
| IPC Callback parameter definition | | |
| Data Fields | | |
| uint32_t | channel | IPC channel. |
| uint32_t | message | Received message data. |
| ipc_event_t | event | Event code. |
| void * | p_context | Context provided to user during callback. |

### ◆ ipc_cfg_t

| struct ipc_cfg_t | |
|---|---|
| IPC Configuration | |
| **Data Fields** | |
| uint32_t | channel |
| | IPC channel. |
| | |
| IRQn_Type | irq |
| | |
| uint8_t | ipl |
| | |

| | |
|---|---|
| void(* | p_callback )(ipc_callback_args_t *p_args) |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |

## Field Documentation

### ◆ irq

IRQn_Type ipc_cfg_t::irq

IPC IRQ number

***Note***

*Only required if the user wants to receive messages or interrupts over IPC. IRQ should be set to FSP_INVALID_VECTOR when not used.*

### ◆ ipl

uint8_t ipc_cfg_t::ipl

IPC Interrupt priority

***Note***

*Only required if the user wants to receive messages or interrupts over IPC.*

### ◆ p_callback

void(* ipc_cfg_t::p_callback) (ipc_callback_args_t *p_args)

Pointer to callback function

***Note***

*Only required if the user wants to receive messages or interrupts over IPC. p_callback should be set to NULL when not used.*

### ◆ ipc_api_t

struct ipc_api_t

Interface definition for IPC

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(ipc_ctrl_t *const p_ctrl, ipc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | messageSend )(ipc_ctrl_t *const p_ctrl, uint32_t message) |
| | |
| fsp_err_t(* | eventGenerate )(ipc_ctrl_t *const p_ctrl, ipc_generate_event_t event) |
| | |

| | |
|---|---|
| fsp_err_t(* | callbackSet )(ipc_ctrl_t *const p_ctrl, void(*p_callback)(ipc_callback_args_t *), void *const p_context, ipc_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(ipc_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* ipc_api_t::open) (ipc_ctrl_t *const p_ctrl, ipc_cfg_t const *const p_cfg)

Open IPC instance.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to the IPC control block. Must be declared by user. Value set here. |
| [in] | p_cfg | Pointer to IPC configuration structure. All elements of this structure must be set by user. |

### ◆ messageSend

fsp_err_t(* ipc_api_t::messageSend) (ipc_ctrl_t *const p_ctrl, uint32_t message)

Send message through the FIFO of the configured IPC channel.

***Note***

*Should only be called from the core on the transmit side of the IPC channel.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IPC control block. |
| [in] | message | Message to send. |

### ◆ eventGenerate

fsp_err_t(* ipc_api_t::eventGenerate) (ipc_ctrl_t *const p_ctrl, ipc_generate_event_t event)

Generate event through configured IPC channel interrupt.

***Note***

*Should only be called from the core on the transmit side of the IPC channel.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IPC control block. |
| [in] | event | Event to generate interrupt for. |

### ◆ callbackSet

fsp_err_t(* ipc_api_t::callbackSet) (ipc_ctrl_t *const p_ctrl, void(*p_callback)(ipc_callback_args_t *), void *const p_context, ipc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IPC control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* ipc_api_t::close) (ipc_ctrl_t *const p_ctrl)

Close IPC instance.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the IPC control block. |

### ◆ ipc_instance_t

struct ipc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

| Data Fields | | |
|---|---|---|
| ipc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| ipc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| ipc_api_t const * | p_api | Pointer to the API structure for this instance. |

### Typedef Documentation

#### ◆ ipc_ctrl_t

| typedef void ipc_ctrl_t |
|---|
| IPC control block. Allocate an instance specific control block to pass into the IPC API calls. |

### Enumeration Type Documentation

#### ◆ ipc_event_t

| enum ipc_event_t | |
|---|---|
| IPC Events | |
| Enumerator | |
| IPC_EVENT_IRQ0 | IRQ Event 0 set. |
| IPC_EVENT_IRQ1 | IRQ Event 1 set. |
| IPC_EVENT_IRQ2 | IRQ Event 2 set. |
| IPC_EVENT_IRQ3 | IRQ Event 3 set. |
| IPC_EVENT_IRQ4 | IRQ Event 4 set. |
| IPC_EVENT_IRQ5 | IRQ Event 5 set. |
| IPC_EVENT_IRQ6 | IRQ Event 6 set. |
| IPC_EVENT_IRQ7 | IRQ Event 7 set. |
| IPC_EVENT_MESSAGE_RECEIVED | Message received over FIFO. |
| IPC_EVENT_FIFO_ERROR_EMPTY | A read was attempted on an empty FIFO. |
| IPC_EVENT_FIFO_ERROR_FULL | Data couldn't be written to a full FIFO. |

### ◆ ipc_generate_event_t

| enum ipc_generate_event_t | |
|---|---|
| IPC Generate Events | |
| Enumerator | |
| IPC_GENERATE_EVENT_IRQ0 | Generate IRQ Event 0. |
| IPC_GENERATE_EVENT_IRQ1 | Generate IRQ Event 1. |
| IPC_GENERATE_EVENT_IRQ2 | Generate IRQ Event 2. |
| IPC_GENERATE_EVENT_IRQ3 | Generate IRQ Event 3. |
| IPC_GENERATE_EVENT_IRQ4 | Generate IRQ Event 4. |
| IPC_GENERATE_EVENT_IRQ5 | Generate IRQ Event 5. |
| IPC_GENERATE_EVENT_IRQ6 | Generate IRQ Event 6. |
| IPC_GENERATE_EVENT_IRQ7 | Generate IRQ Event 7. |

## 5.3.17 Timers

Interfaces

### Detailed Description

Timers Interfaces.

### Modules

| | |
|---|---|
| POEG Interface | Interface for the Port Output Enable for GPT. |
| RTC Interface | Interface for accessing the Realtime Clock. |
| Three-Phase Interface | Interface for three-phase timer functions. |

| | |
|---|---|
| | Timer Interface<br>Interface for timer functions. |

### 5.3.17.1 POEG Interface

Interfaces » Timers

#### Detailed Description

Interface for the Port Output Enable for GPT.

Defines the API and data structures for the Port Output Enable for GPT (POEG) interface.

## Summary

The POEG disables GPT output pins based on configurable events.

#### Data Structures

| | |
|---|---|
| struct | poeg_status_t |
| struct | poeg_callback_args_t |
| struct | poeg_cfg_t |
| struct | poeg_api_t |
| struct | poeg_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | poeg_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | poeg_state_t |
| enum | poeg_trigger_t |
| enum | poeg_gtetrg_polarity_t |
| enum | poeg_gtetrg_noise_filter_t |

#### Data Structure Documentation

◆ **poeg_status_t**

| struct poeg_status_t | | |
|---|---|---|
| POEG status | | |
| Data Fields | | |
| poeg_state_t | state | Current state of POEG. |

### ◆ poeg_callback_args_t

| struct poeg_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data, set in poeg_cfg_t. |

### ◆ poeg_cfg_t

| struct poeg_cfg_t | |
|---|---|
| User configuration structure, used in the open function. | |
| **Data Fields** | |
| poeg_trigger_t | trigger |
| | Select one or more triggers for the POEG. |
| | |
| poeg_gtetrg_polarity_t | polarity |
| | Select the polarity for the GTETRG pin. |
| | |
| poeg_gtetrg_noise_filter_t | noise_filter |
| | Configure the GTETRG noise filter. |
| | |
| void(* | p_callback )(poeg_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| uint32_t | unit |
| | POEG unit to be used. |
| | |
| uint32_t | channel |

| | |
|---|---|
| | Channel 0 corresponds to GTETRGA, 1 to GTETRGB, etc. |
| | |
| IRQn_Type | irq |
| | Interrupt number assigned to this instance. |
| | |
| uint8_t | ipl |
| | POEG interrupt priority. |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

## Field Documentation

### ◆ p_callback

void(* poeg_cfg_t::p_callback) (poeg_callback_args_t *p_args)

Callback called when a POEG interrupt occurs.

### ◆ p_context

void* poeg_cfg_t::p_context

Placeholder for user data. Passed to the user callback in poeg_callback_args_t.

### ◆ poeg_api_t

struct poeg_api_t

Port Output Enable for GPT (POEG) API structure. POEG functions implemented at the HAL layer will follow this API.

**Data Fields**

| | |
|---|---|
| fsp_err_t(* | open )(poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | statusGet )(poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status) |
| | |
| fsp_err_t(* | callbackSet )(poeg_ctrl_t *const p_ctrl,<br>void(*p_callback)(poeg_callback_args_t *), void *const p_context,<br>poeg_callback_args_t *const p_callback_memory) |
| | |

| | |
|---|---|
| fsp_err_t(* | outputDisable )(poeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(poeg_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | close )(poeg_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* poeg_api_t::open) (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ statusGet

fsp_err_t(* poeg_api_t::statusGet) (poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)

Gets the current driver state.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |
| [out] | p_status | Provides the current state of the POEG. |

### ◆ callbackSet

fsp_err_t(* poeg_api_t::callbackSet) (poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void *const p_context, poeg_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call for this timer. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ outputDisable

fsp_err_t(* poeg_api_t::outputDisable) (poeg_ctrl_t *const p_ctrl)

Disables GPT output pins by software request.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |

### ◆ reset

fsp_err_t(* poeg_api_t::reset) (poeg_ctrl_t *const p_ctrl)

Attempts to clear status flags to reenable GPT output pins. Confirm all status flags are cleared after calling this function by calling poeg_api_t::statusGet().

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in poeg_api_t::open call. |

### ◆ close

| fsp_err_t(* poeg_api_t::close) (poeg_ctrl_t *const p_ctrl) | | |
|---|---|---|
| Disables POEG interrupt.<br><br>**Parameters**<br>[in] p_ctrl — Control block set in poeg_api_t::open call. | | |

### ◆ poeg_instance_t

| struct poeg_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| poeg_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| poeg_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| poeg_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ poeg_ctrl_t

| typedef void poeg_ctrl_t |
|---|
| POEG control block. Allocate an instance specific control block to pass into the POEG API calls. |

## Enumeration Type Documentation

### ◆ poeg_state_t

| enum poeg_state_t | |
|---|---|
| POEG states. | |
| Enumerator | |
| POEG_STATE_NO_DISABLE_REQUEST | GPT output is not disabled by POEG. |
| POEG_STATE_PIN_DISABLE_REQUEST | GPT output disabled due to GTETRG pin level. |
| POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST | GPT output disabled due to high speed analog comparator or GPT. |
| POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST | GPT output disabled due to main oscillator stop. |
| POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST | GPT output disabled due to poeg_api_t::outputDisable() |
| POEG_STATE_DSMIF0_DISABLE_REQUEST | GPT output disabled due to DSMIF0. |
| POEG_STATE_DSMIF1_DISABLE_REQUEST | GPT output disabled due to DSMIF1. |
| POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE | GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input. |
| POEG_STATE_NO_DISABLE_REQUEST | GPT output is not disabled by POEG. |
| POEG_STATE_PIN_DISABLE_REQUEST | GPT output disabled due to GTETRG pin level. |
| POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST | GPT output disabled due to high speed analog comparator or GPT. |
| POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST | GPT output disabled due to main oscillator stop. |
| POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST | GPT output disabled due to poeg_api_t::outputDisable() |
| POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE | GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input. |

### ◆ poeg_trigger_t

| enum poeg_trigger_t | |
|---|---|
| Triggers that will disable GPT output pins. | |
| Enumerator | |
| POEG_TRIGGER_SOFTWARE | Software disable is always supported with POEG. Select this option if no other triggers are used. |
| POEG_TRIGGER_PIN | Disable GPT output based on GTETRG input level. |
| POEG_TRIGGER_GPT_OUTPUT_LEVEL | Disable GPT output based on GPT output pin levels. |
| POEG_TRIGGER_OSCILLATION_STOP | Disable GPT output based on main oscillator stop. |
| POEG_TRIGGER_ACMPHS0 | Disable GPT output based on ACMPHS0 comparator result. |
| POEG_TRIGGER_ACMPHS1 | Disable GPT output based on ACMPHS1 comparator result. |
| POEG_TRIGGER_ACMPHS2 | Disable GPT output based on ACMPHS2 comparator result. |
| POEG_TRIGGER_ACMPHS3 | Disable GPT output based on ACMPHS3 comparator result. |
| POEG_TRIGGER_ACMPHS4 | Disable GPT output based on ACMPHS4 comparator result. |
| POEG_TRIGGER_ACMPHS5 | Disable GPT output based on ACMPHS5 comparator result. |
| POEG_TRIGGER_DSMIF0 | Disable GPT output based on DSMIF0 overcurrent detection window notification. |
| POEG_TRIGGER_DSMIF1 | Disable GPT output based on DSMIF1 overcurrent detection window notification. |
| POEG_TRIGGER_SOFTWARE | Software disable is always supported with POEG. Select this option if no other triggers are used. |
| POEG_TRIGGER_PIN | Disable GPT output based on GTETRG input level. |

| | |
|---|---|
| POEG_TRIGGER_GPT_OUTPUT_LEVEL | Disable GPT output based on GPT output pin levels. |
| POEG_TRIGGER_OSCILLATION_STOP | Disable GPT output based on main oscillator stop. |
| POEG_TRIGGER_ACMPHS0 | Disable GPT output based on ACMPHS0 comparator result. |
| POEG_TRIGGER_ACMPHS1 | Disable GPT output based on ACMPHS1 comparator result. |
| POEG_TRIGGER_ACMPHS2 | Disable GPT output based on ACMPHS2 comparator result. |
| POEG_TRIGGER_ACMPHS3 | Disable GPT output based on ACMPHS3 comparator result. |
| POEG_TRIGGER_ACMPHS4 | Disable GPT output based on ACMPHS4 comparator result. |
| POEG_TRIGGER_ACMPHS5 | Disable GPT output based on ACMPHS5 comparator result. |

### ◆ poeg_gtetrg_polarity_t

| enum poeg_gtetrg_polarity_t | |
|---|---|
| GTETRG polarity. | |
| Enumerator | |
| POEG_GTETRG_POLARITY_ACTIVE_HIGH | Disable GPT output based when GTETRG input level is high. |
| POEG_GTETRG_POLARITY_ACTIVE_LOW | Disable GPT output based when GTETRG input level is low. |

### ◆ poeg_gtetrg_noise_filter_t

| enum poeg_gtetrg_noise_filter_t | |
|---|---|
| GTETRG noise filter. For the input signal to pass through the noise filter, the active level set in poeg_gtetrg_polarity_t must be read 3 consecutive times at the sampling clock selected. | |
| Enumerator | |
| POEG_GTETRG_NOISE_FILTER_DISABLED | No noise filter applied to GTETRG input. |
| POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_1 | Apply noise filter with sample clock equal to Clock source/1. |
| POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_8 | Apply noise filter with sample clock equal to Clock source/8. |
| POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_32 | Apply noise filter with sample clock equal to Clock source/32. |
| POEG_GTETRG_NOISE_FILTER_CLK_SOURCE_DIV_128 | Apply noise filter with sample clock equal to Clock source/128. |

## 5.3.17.2 RTC Interface

Interfaces » Timers

### Detailed Description

Interface for accessing the Realtime Clock.

# Summary

The RTC Interface is for configuring Real Time Clock (RTC) functionality including alarm, periodic notification and error adjustment.

### Data Structures

| | |
|---|---|
| struct | rtc_callback_args_t |
| struct | rtc_error_adjustment_cfg_t |
| struct | rtc_alarm_time_t |
| struct | rtc_time_capture_t |
| struct | rtc_info_t |

| | |
|---|---|
| struct | rtc_cfg_t |
| struct | rtc_api_t |
| struct | rtc_instance_t |

### Typedefs

| | |
|---|---|
| typedef struct tm | rtc_time_t |
| typedef void | rtc_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | rtc_event_t |
| enum | rtc_alarm_channel_t |
| enum | rtc_clock_source_t |
| enum | rtc_status_t |
| enum | rtc_error_adjustment_t |
| enum | rtc_error_adjustment_mode_t |
| enum | rtc_error_adjustment_period_t |
| enum | rtc_periodic_irq_select_t |
| enum | rtc_time_capture_source_t |
| enum | rtc_time_capture_mode_t |
| enum | rtc_time_capture_noise_filter_t |

## Data Structure Documentation

### ◆ rtc_callback_args_t

| struct rtc_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| rtc_event_t | event | The event can be used to identify what caused the callback (compare match or error). |
| void * | p_context | Placeholder for user data. |

### ◆ rtc_error_adjustment_cfg_t

| struct rtc_error_adjustment_cfg_t | | |
|---|---|---|
| Time error adjustment value configuration | | |
| Data Fields | | |
| rtc_error_adjustment_mode_t | adjustment_mode | Automatic Adjustment Enable/Disable. |
| rtc_error_adjustment_period_t | adjustment_period | Error Adjustment period. |
| rtc_error_adjustment_t | adjustment_type | Time error adjustment setting. |
| uint32_t | adjustment_value | Value of the prescaler for error adjustment. |

### ◆ rtc_alarm_time_t

| struct rtc_alarm_time_t | | |
|---|---|---|
| Alarm time setting structure | | |
| Data Fields | | |
| int | time_minute | Time structure. |
| int | time_hour | |
| union rtc_alarm_time_t | __unnamed__ | |
| union rtc_alarm_time_t | __unnamed__ | |
| union rtc_alarm_time_t | __unnamed__ | |
| rtc_time_t | time | Time structure. |
| bool | sec_match | Enable the alarm based on a match of the seconds field. |
| bool | min_match | Enable the alarm based on a match of the minutes field. |
| bool | hour_match | Enable the alarm based on a match of the hours field. |
| bool | mday_match | Enable the alarm based on a match of the days field. |
| bool | mon_match | Enable the alarm based on a match of the months field. |
| bool | year_match | Enable the alarm based on a match of the years field. |
| bool | dayofweek_match | Enable the alarm based on a match of the dayofweek field. |
| bool | sunday_match | Enable the alarm on Sunday. |
| bool | monday_match | Enable the alarm on Monday. |
| bool | tuesday_match | Enable the alarm on Tuesday. |

| | | |
|---|---|---|
| bool | wednesday_match | Enable the alarm on Wednesday. |
| bool | thursday_match | Enable the alarm on Thursday. |
| bool | friday_match | Enable the alarm on Friday. |
| bool | saturday_match | Enable the alarm on Saturday. |
| rtc_alarm_channel_t | channel | Select alarm 0 or alarm 1. |

### ◆ rtc_time_capture_t

| struct rtc_time_capture_t | | |
|---|---|---|
| Time capture configuration structure | | |
| Data Fields | | |
| rtc_time_t | time | Time structure. |
| uint8_t | channel | Capture channel. |
| rtc_time_capture_source_t | source | Trigger source. |
| rtc_time_capture_noise_filter_t | noise_filter | Noise filter. |
| rtc_time_capture_mode_t | mode | Capture mode. |

### ◆ rtc_info_t

| struct rtc_info_t | | |
|---|---|---|
| RTC Information Structure for information returned by infoGet() | | |
| Data Fields | | |
| rtc_clock_source_t | clock_source | Clock source for the RTC block. |
| rtc_status_t | status | RTC run status. |

### ◆ rtc_cfg_t

| struct rtc_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| rtc_clock_source_t | clock_source |
| | Clock source for the RTC block. |
| | |
| uint32_t | freq_compare_value |
| | The frequency comparison value. |
| | |
| rtc_error_adjustment_cfg_t | p_err_cfg |

| | |
|---|---|
| const *const | |
| | Pointer to Error Adjustment configuration. |
| | |
| uint8_t | alarm_ipl |
| | Alarm interrupt priority. |
| | |
| IRQn_Type | alarm_irq |
| | Alarm interrupt vector. |
| | |
| uint8_t | periodic_ipl |
| | Periodic interrupt priority. |
| | |
| IRQn_Type | periodic_irq |
| | Periodic interrupt vector. |
| | |
| uint8_t | carry_ipl |
| | Carry interrupt priority. |
| | |
| IRQn_Type | carry_irq |
| | Carry interrupt vector. |
| | |
| void(* | p_callback )(rtc_callback_args_t *p_args) |
| | Called from the ISR. |
| | |
| void * | p_context |
| | User defined context passed into callback function. |
| | |
| void const * | p_extend |

| | |
|---|---|
| | RTC hardware dependant configuration. |
| | |

### ◆ rtc_api_t

| struct rtc_api_t | |
|---|---|
| RTC driver structure. General RTC functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | close )(rtc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | clockSourceSet )(rtc_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | calendarTimeSet )(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| | |
| fsp_err_t(* | calendarTimeGet )(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time) |
| | |
| fsp_err_t(* | calendarAlarmSet )(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| | |
| fsp_err_t(* | calendarAlarmGet )(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm) |
| | |
| fsp_err_t(* | periodicIrqRateSet )(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate) |
| | |
| fsp_err_t(* | errorAdjustmentSet )(rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg) |
| | |
| fsp_err_t(* | callbackSet )(rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | infoGet )(rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info) |
| | |
| fsp_err_t(* | timeCaptureSet )(rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const |

| | |
|---|---|
| | p_time_capture) |
| | |
| fsp_err_t(* | timeCaptureGet )(rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)

Open the RTC driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_cfg | Pointer to the configuration structure |

### ◆ close

fsp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)

Close the RTC driver.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle. |

### ◆ clockSourceSet

fsp_err_t(* rtc_api_t::clockSourceSet) (rtc_ctrl_t *const p_ctrl)

Sets the RTC clock source.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |

### ◆ calendarTimeSet

fsp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Set the calendar time and start the calendar counter

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_time | Pointer to a time structure that contains the time to set |

### ◆ calendarTimeGet

fsp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Get the calendar time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [out] | p_time | Pointer to a time structure that contains the time to get |

### ◆ calendarAlarmSet

fsp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Set the calendar alarm time and enable the alarm interrupt.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_alarm | Pointer to an alarm structure that contains the alarm time to set |

### ◆ calendarAlarmGet

fsp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Get the calendar alarm time.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [out] | p_alarm | Pointer to an alarm structure to fill up with the alarm time |

### ◆ periodicIrqRateSet

fsp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)

Set the periodic irq rate

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | rate | Rate of periodic interrupts |

### ◆ errorAdjustmentSet

fsp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)

Set time error adjustment.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [in] | err_adj_cfg | Pointer to the Error Adjustment Config |

### ◆ callbackSet

fsp_err_t(* rtc_api_t::callbackSet) (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void *const p_context, rtc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to the RTC control block. |
| [in] | p_callback | Callback function |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated |

### ◆ infoGet

fsp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)

Return the currently configure clock source for the RTC

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control handle structure |
| [out] | p_rtc_info | Pointer to RTC information structure |

### ◆ timeCaptureSet

fsp_err_t(* rtc_api_t::timeCaptureSet) (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Config Time capture

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [in] | p_time_capture | Pointer to a time capture structure that contains the configuration |

### ◆ timeCaptureGet

fsp_err_t(* rtc_api_t::timeCaptureGet) (rtc_ctrl_t *const p_ctrl, rtc_time_capture_t *const p_time_capture)

Get the capture time and clear bit status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to RTC device handle |
| [out] | p_time_capture | Pointer to a time capture structure to fill up with the time capture |

### ◆ rtc_instance_t

struct rtc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| rtc_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| rtc_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |

| | | |
|---|---|---|
| rtc_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ rtc_time_t

| typedef struct tm rtc_time_t |
|---|
| Date and time structure defined in C standard library <time.h> |

### ◆ rtc_ctrl_t

| typedef void rtc_ctrl_t |
|---|
| RTC control block. Allocate an instance specific control block to pass into the RTC API calls. |

## Enumeration Type Documentation

### ◆ rtc_event_t

| enum rtc_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| RTC_EVENT_ALARM_IRQ | Real Time Clock ALARM 0 IRQ. |
| RTC_EVENT_ALARM1_IRQ | Real Time Clock ALARM 1 IRQ. |
| RTC_EVENT_PERIODIC_IRQ | Real Time Clock PERIODIC IRQ. |

### ◆ rtc_alarm_channel_t

| enum rtc_alarm_channel_t |
|---|
| RTC alarm channel |

### ◆ rtc_clock_source_t

| enum rtc_clock_source_t | |
|---|---|
| Clock source for the RTC block | |
| Enumerator | |
| RTC_CLOCK_SOURCE_SUBCLK | Sub-clock oscillator. |
| RTC_CLOCK_SOURCE_LOCO | Low power On Chip Oscillator. |
| RTC_CLOCK_SOURCE_MAINCLK | Main clock oscillator. |

### ◆ rtc_status_t

| enum rtc_status_t | |
|---|---|
| RTC run state | |
| Enumerator | |
| RTC_STATUS_STOPPED | RTC counter is stopped. |
| RTC_STATUS_RUNNING | RTC counter is running. |

### ◆ rtc_error_adjustment_t

| enum rtc_error_adjustment_t | |
|---|---|
| Time error adjustment settings | |
| Enumerator | |
| RTC_ERROR_ADJUSTMENT_NONE | Adjustment is not performed. |
| RTC_ERROR_ADJUSTMENT_ADD_PRESCALER | Adjustment is performed by the addition to the prescaler. |
| RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER | Adjustment is performed by the subtraction from the prescaler. |

### ◆ rtc_error_adjustment_mode_t

| enum rtc_error_adjustment_mode_t | |
|---|---|
| Time error adjustment mode settings | |
| Enumerator | |
| RTC_ERROR_ADJUSTMENT_MODE_MANUAL | Adjustment mode is set to manual. |
| RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC | Adjustment mode is set to automatic. |

### ◆ rtc_error_adjustment_period_t

| enum rtc_error_adjustment_period_t | |
|---|---|
| Time error adjustment period settings | |
| Enumerator | |
| RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE | Adjustment period is set to every one minute. |
| RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND | Adjustment period is set to every ten second. |
| RTC_ERROR_ADJUSTMENT_PERIOD_NONE | Adjustment period not supported in manual mode. |
| RTC_ERROR_ADJUSTMENT_PERIOD_20_SECOND | Adjustment period is set to every twenty seconds. |

### ◆ rtc_periodic_irq_select_t

| enum rtc_periodic_irq_select_t | |
|---|---|
| Periodic Interrupt select | |
| Enumerator | |
| RTC_PERIODIC_IRQ_SELECT_NONE | A periodic irq is not generated. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND | A periodic irq is generated every 1/2 second. |
| RTC_PERIODIC_IRQ_SELECT_1_SECOND | A periodic irq is generated every 1 second. |
| RTC_PERIODIC_IRQ_SELECT_1_MINUTE | A periodic irq is generated every 1 minute. |
| RTC_PERIODIC_IRQ_SELECT_1_HOUR | A periodic irq is generated every 1 hour. |
| RTC_PERIODIC_IRQ_SELECT_1_DAY | A periodic irq is generated every 1 day. |
| RTC_PERIODIC_IRQ_SELECT_1_MONTH | A periodic irq is generated every 1 month. |
| RTC_PERIODIC_IRQ_SELECT_NONE | A periodic irq is not generated. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND | A periodic irq is generated every 1/2 second. |
| RTC_PERIODIC_IRQ_SELECT_1_SECOND | A periodic irq is generated every 1 second. |
| RTC_PERIODIC_IRQ_SELECT_1_MINUTE | A periodic irq is generated every 1 minute. |
| RTC_PERIODIC_IRQ_SELECT_1_HOUR | A periodic irq is generated every 1 hour. |
| RTC_PERIODIC_IRQ_SELECT_1_DAY | A periodic irq is generated every 1 day. |
| RTC_PERIODIC_IRQ_SELECT_1_MONTH | A periodic irq is generated every 1 month. |
| RTC_PERIODIC_IRQ_SELECT_NONE | A periodic irq is not generated. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND | A periodic irq is generated every 1/2 second. |
| RTC_PERIODIC_IRQ_SELECT_1_SECOND | A periodic irq is generated every 1 second. |
| RTC_PERIODIC_IRQ_SELECT_1_MINUTE | A periodic irq is generated every 1 minute. |
| RTC_PERIODIC_IRQ_SELECT_1_HOUR | A periodic irq is generated every 1 hour. |
| RTC_PERIODIC_IRQ_SELECT_1_DAY | A periodic irq is generated every 1 day. |

| | |
|---|---|
| RTC_PERIODIC_IRQ_SELECT_1_MONTH | A periodic irq is generated every 1 month. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND | A periodic irq is generated every 1/256 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND | A periodic irq is generated every 1/128 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND | A periodic irq is generated every 1/64 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND | A periodic irq is generated every 1/32 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND | A periodic irq is generated every 1/16 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND | A periodic irq is generated every 1/8 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND | A periodic irq is generated every 1/4 second. |
| RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND | A periodic irq is generated every 1/2 second. |
| RTC_PERIODIC_IRQ_SELECT_1_SECOND | A periodic irq is generated every 1 second. |
| RTC_PERIODIC_IRQ_SELECT_2_SECOND | A periodic irq is generated every 2 seconds. |
| RTC_PERIODIC_IRQ_SELECT_1_MINUTE | A periodic irq is generated every 1 minute. |
| RTC_PERIODIC_IRQ_SELECT_1_HOUR | A periodic irq is generated every 1 hour. |
| RTC_PERIODIC_IRQ_SELECT_1_DAY | A periodic irq is generated every 1 day. |
| RTC_PERIODIC_IRQ_SELECT_1_MONTH | A periodic irq is generated every 1 month. |

### ◆ rtc_time_capture_source_t

| enum rtc_time_capture_source_t | |
|---|---|
| Time capture trigger source | |
| Enumerator | |
| RTC_TIME_CAPTURE_SOURCE_DISABLED | Disable trigger. |
| RTC_TIME_CAPTURE_SOURCE_PIN_RISING | Rising edge pin trigger. |
| RTC_TIME_CAPTURE_SOURCE_PIN_FALLING | Falling edge pin trigger. |
| RTC_TIME_CAPTURE_SOURCE_PIN_BOTH | Both edges pin trigger. |
| RTC_TIME_CAPTURE_SOURCE_SOFTWARE | Software trigger. |
| RTC_TIME_CAPTURE_SOURCE_ELC_EVENT | ELC event trigger. |

### ◆ rtc_time_capture_mode_t

| enum rtc_time_capture_mode_t | |
|---|---|
| Time capture trigger mode | |
| Enumerator | |
| RTC_TIME_CAPTURE_MODE_CONTINUOUS | Continuous capturing to all capturing channels. |
| RTC_TIME_CAPTURE_MODE_ONE_SHOT | Single capture to a particular channel. |

### ◆ rtc_time_capture_noise_filter_t

| enum rtc_time_capture_noise_filter_t | |
|---|---|
| Time capture noise filter control | |
| Enumerator | |
| RTC_TIME_CAPTURE_NOISE_FILTER_OFF | Turn noise filter off. |
| RTC_TIME_CAPTURE_NOISE_FILTER_ON | Turn noise filter on (count source) |
| RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_32 | Turn noise filter on (count source by divided by 32) |
| RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_4096 | Turn noise filter on (count source by divided by 4096) |
| RTC_TIME_CAPTURE_NOISE_FILTER_ON_DIVIDER_8192 | Turn noise filter on (count source by divided by 8192) |

### 5.3.17.3 Three-Phase Interface

Interfaces » Timers

#### Detailed Description

Interface for three-phase timer functions.

# Summary

The Three-Phase interface provides functionality for synchronous start/stop/reset control of three timer channels for use in 3-phase motor control applications.

#### Data Structures

| | |
|---|---|
| struct | three_phase_duty_cycle_t |
| struct | three_phase_cfg_t |
| struct | three_phase_api_t |
| struct | three_phase_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | three_phase_ctrl_t |

## Enumerations

| | |
|---|---|
| enum | three_phase_channel_t |
| enum | three_phase_buffer_mode_t |

## Data Structure Documentation

### ◆ three_phase_duty_cycle_t

| struct three_phase_duty_cycle_t | | |
|---|---|---|
| Struct for passing duty cycle values to three_phase_api_t::dutyCycleSet | | |
| Data Fields | | |
| uint32_t | duty[3] | Duty cycle. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transferred to GTCCRA/B at the trough. |
| uint32_t | duty_buffer[3] | Double-buffer for duty cycle values. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transferred to GTCCRA/B at the crest. |

### ◆ three_phase_cfg_t

| struct three_phase_cfg_t | | |
|---|---|---|
| User configuration structure, used in open function | | |
| Data Fields | | |
| three_phase_buffer_mode_t | buffer_mode | Single or double-buffer mode. |
| timer_instance_t const * | p_timer_instance[3] | Pointer to the timer instance structs. |
| three_phase_channel_t | callback_ch | Channel to enable callback when using three_phase_api_t::callbackSet. |
| uint32_t | channel_mask | Bitmask of timer channels used by this module. |
| void * | p_context | Placeholder for user data. Passed to the user callback in timer_callback_args_t. |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ three_phase_api_t

| struct three_phase_api_t | |
|---|---|
| Three-Phase API structure. | |
| **Data Fields** | |
| fsp_err_t(* | open )(three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(three_phase_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | dutyCycleSet )(three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle) |
| | |
| fsp_err_t(* | callbackSet )(three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |
| | |
| fsp_err_t(* | close )(three_phase_ctrl_t *const p_ctrl) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* three_phase_api_t::open) (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ start

fsp_err_t(* three_phase_api_t::start) (three_phase_ctrl_t *const p_ctrl)

Start all three timers synchronously.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ stop

fsp_err_t(* three_phase_api_t::stop) (three_phase_ctrl_t *const p_ctrl)

Stop all three timers synchronously.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ reset

fsp_err_t(* three_phase_api_t::reset) (three_phase_ctrl_t *const p_ctrl)

Reset all three timers synchronously.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ dutyCycleSet

fsp_err_t(* three_phase_api_t::dutyCycleSet) (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)

Sets the duty cycle match values. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |
| [in] | p_duty_cycle | Duty cycle values for all three timer channels. |

### ◆ callbackSet

fsp_err_t(* three_phase_api_t::callbackSet) (three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call. |
| [in] | p_callback | Callback function to register with GPT U-channel |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* three_phase_api_t::close) (three_phase_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in three_phase_api_t::open call for this timer. |

### ◆ three_phase_instance_t

struct three_phase_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

| | | |
|---|---|---|
| three_phase_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| three_phase_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| three_phase_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ three_phase_ctrl_t

| typedef void three_phase_ctrl_t |
| --- |
| Three-Phase control block. Allocate an instance specific control block to pass into the timer API calls. |

#### Enumeration Type Documentation

### ◆ three_phase_channel_t

| enum three_phase_channel_t | |
| --- | --- |
| Timer channel indices | |
| Enumerator | |
| THREE_PHASE_CHANNEL_U | U-channel index. |
| THREE_PHASE_CHANNEL_V | V-channel index. |
| THREE_PHASE_CHANNEL_W | W-channel index. |

### ◆ three_phase_buffer_mode_t

| enum three_phase_buffer_mode_t | |
| --- | --- |
| Buffering mode | |
| Enumerator | |
| THREE_PHASE_BUFFER_MODE_SINGLE | Single-buffer mode. |
| THREE_PHASE_BUFFER_MODE_DOUBLE | Double-buffer mode. |

## 5.3.17.4 Timer Interface

Interfaces » Timers

#### Detailed Description

Interface for timer functions.

# Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, PWM output, and free-running timer mode. After each timer cycle (overflow or underflow), an

interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration timer_on_<instance>_cfg_t defined in r_<instance>.h.

### Data Structures

| | |
|---|---|
| struct | timer_callback_args_t |
| struct | timer_info_t |
| struct | timer_status_t |
| struct | timer_cfg_t |
| struct | timer_api_t |
| struct | timer_instance_t |

### Typedefs

| | |
|---|---|
| typedef void | timer_ctrl_t |

### Enumerations

| | |
|---|---|
| enum | timer_event_t |
| enum | timer_variant_t |
| enum | timer_compare_match_t |
| enum | timer_state_t |
| enum | timer_mode_t |
| enum | timer_direction_t |
| enum | timer_source_div_t |

## Data Structure Documentation

### ◆ timer_callback_args_t

| struct timer_callback_args_t | | |
|---|---|---|
| Callback function parameter data | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t. |
| | | |

| | | |
|---|---|---|
| timer_event_t | event | The event can be used to identify what caused the callback. |
| uint32_t | capture | Most recent capture, only valid if event is TIMER_EVENT_CAPTURE_A or TIMER_EVENT_CAPTURE_B. |

### ◆ timer_info_t

| struct timer_info_t | | |
|---|---|---|
| Timer information structure to store various information for a timer resource | | |
| Data Fields | | |
| timer_direction_t | count_direction | Clock counting direction of the timer. |
| uint32_t | clock_frequency | Clock frequency of the timer counter. |
| uint32_t | period_counts | Period in raw timer counts.<br>*Note*<br>*For triangle wave PWM modes, the full period is double this value.* |

### ◆ timer_status_t

| struct timer_status_t | | |
|---|---|---|
| Current timer status. | | |
| Data Fields | | |
| uint32_t | counter | Current counter value. |
| timer_state_t | state | Current timer state (running or stopped) |

### ◆ timer_cfg_t

| struct timer_cfg_t | |
|---|---|
| User configuration structure, used in open function | |
| **Data Fields** | |
| timer_mode_t | mode |
| | Select enumerated value from timer_mode_t. |
| | |
| uint32_t | period_counts |
| | Period in raw timer counts. |

| | |
|---|---|
| | |
| timer_source_div_t | source_div |
| | Source clock divider. |
| | |
| uint32_t | duty_cycle_counts |
| | Duty cycle in counts. |
| | |
| uint8_t | channel |
| | |
| uint8_t | cycle_end_ipl |
| | Cycle end interrupt priority. |
| | |
| IRQn_Type | cycle_end_irq |
| | Cycle end interrupt. |
| | |
| void(* | p_callback )(timer_callback_args_t *p_args) |
| | |
| void * | p_context |
| | |
| void const * | p_extend |
| | Extension parameter for hardware specific settings. |
| | |

## Field Documentation

### ◆ channel

uint8_t timer_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

### ◆ p_callback

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

### ◆ p_context

| void* timer_cfg_t::p_context |
|---|
| Placeholder for user data. Passed to the user callback in timer_callback_args_t. |

### ◆ timer_api_t

| struct timer_api_t | |
|---|---|
| Timer API structure. General timer functions implemented at the HAL layer follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | start )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | stop )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reset )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | enable )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(timer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | periodSet )(timer_ctrl_t *const p_ctrl, uint32_t const period) |
| | |
| fsp_err_t(* | dutyCycleSet )(timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin) |
| | |
| fsp_err_t(* | compareMatchSet )(timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel) |
| | |
| fsp_err_t(* | infoGet )(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info) |
| | |
| fsp_err_t(* | statusGet )(timer_ctrl_t *const p_ctrl, timer_status_t *const p_status) |
| | |
| fsp_err_t(* | callbackSet )(timer_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory) |

| | |
|---|---|
| fsp_err_t(* | close )(timer_ctrl_t *const p_ctrl) |

## Field Documentation

### ◆ open

fsp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ start

fsp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)

Start the counter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ stop

fsp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)

Stop the counter.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ reset

fsp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)

Reset the counter to the initial value.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ enable

fsp_err_t(* timer_api_t::enable) (timer_ctrl_t *const p_ctrl)

Enables input capture.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ disable

fsp_err_t(* timer_api_t::disable) (timer_ctrl_t *const p_ctrl)

Disables input capture.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ periodSet

fsp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, uint32_t const period)

Set the time until the timer expires. See implementation for details of period update timing.

***Note***

*Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in timer_api_t::open.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | period | Time until timer should expire. |

### ◆ dutyCycleSet

fsp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)

Sets the number of counts for the pin level to be high. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | duty_cycle_counts | Time until duty cycle should expire. |
| [in] | pin | Which output pin to update. See implementation for details. |

### ◆ compareMatchSet

fsp_err_t(* timer_api_t::compareMatchSet) (timer_ctrl_t *const p_ctrl, uint32_t const compare_match_value, timer_compare_match_t const match_channel)

Set a compare match value in raw counts.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | compare_match_value | Timer value to trigger a compare match event. |
| [in] | match_channel | Which channel to update. |

### ◆ infoGet

fsp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Stores timer information in p_info.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [out] | p_info | Collection of information for this timer. |

### ◆ statusGet

fsp_err_t(* timer_api_t::statusGet) (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

Get the current counter value and timer state and store it in p_status.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [out] | p_status | Current status of this timer. |

### ◆ callbackSet

fsp_err_t(* timer_api_t::callbackSet) (timer_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_working_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ close

fsp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in timer_api_t::open call for this timer. |

### ◆ timer_instance_t

struct timer_instance_t

| This structure encompasses everything that is needed to use an instance of this interface. | | |
|---|---|---|
| Data Fields | | |
| timer_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| timer_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| timer_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ timer_ctrl_t

| typedef void timer_ctrl_t |
|---|
| Timer control block. Allocate an instance specific control block to pass into the timer API calls. |

## Enumeration Type Documentation

### ◆ timer_event_t

| enum timer_event_t | |
|---|---|
| Events that can trigger a callback function | |
| Enumerator | |
| TIMER_EVENT_CYCLE_END | Requested timer delay has expired or timer has wrapped around. |
| TIMER_EVENT_HIGHER_8BIT_CYCLE_END | Requested higher 8-bit timer has expired. |
| TIMER_EVENT_CAPTURE_EDGE | A capture has occurred when detecting edge. |
| TIMER_EVENT_SLAVE_CYCLE_END | Requested timer pulse width (One-shot) or duty cycle (PWM) has expired. |
| TIMER_EVENT_MASTER_CYCLE_END | Requested timer delay (One-shot) or period (PWM) has expired. |
| TIMER_EVENT_CYCLE_END | Requested timer delay has expired or timer has wrapped around. |
| TIMER_EVENT_HIGHER_8BIT_CYCLE_END | Requested higher 8-bit timer has expired. |
| TIMER_EVENT_CAPTURE_EDGE | A capture has occurred when detecting edge. |
| TIMER_EVENT_SLAVE_CYCLE_END | Requested timer pulse width (One-shot) or |

| | |
|---|---|
| | duty cycle (PWM) has expired. |
| TIMER_EVENT_MASTER_CYCLE_END | Requested timer delay (One-shot) or period (PWM) has expired. |
| TIMER_EVENT_CYCLE_END | Requested timer delay has expired or timer has wrapped around. |
| TIMER_EVENT_HIGHER_8BIT_CYCLE_END | Requested higher 8-bit timer has expired. |
| TIMER_EVENT_CAPTURE_EDGE | A capture has occurred when detecting edge. |
| TIMER_EVENT_SLAVE_CYCLE_END | Requested timer pulse width (One-shot) or duty cycle (PWM) has expired. |
| TIMER_EVENT_MASTER_CYCLE_END | Requested timer delay (One-shot) or period (PWM) has expired. |
| TIMER_EVENT_CYCLE_END | Requested timer delay has expired or timer has wrapped around. |
| TIMER_EVENT_CREST | Timer crest event (counter is at a maximum, triangle-wave PWM only) |
| TIMER_EVENT_CAPTURE_A | A capture has occurred on signal A. |
| TIMER_EVENT_CAPTURE_B | A capture has occurred on signal B. |
| TIMER_EVENT_TROUGH | Timer trough event (counter is 0, triangle-wave PWM only. |
| TIMER_EVENT_COMPARE_A | A compare has occurred on signal A. |
| TIMER_EVENT_COMPARE_B | A compare has occurred on signal B. |
| TIMER_EVENT_COMPARE_C | A compare has occurred on signal C. |
| TIMER_EVENT_COMPARE_D | A compare has occurred on signal D. |
| TIMER_EVENT_COMPARE_E | A compare has occurred on signal E. |
| TIMER_EVENT_COMPARE_F | A compare has occurred on signal F. |
| TIMER_EVENT_DEAD_TIME | Dead time event. |

### ◆ timer_variant_t

| enum timer_variant_t | |
|---|---|
| Timer variant types. | |
| Enumerator | |
| TIMER_VARIANT_32_BIT | 32-bit timer |
| TIMER_VARIANT_16_BIT | 16-bit timer |

### ◆ timer_compare_match_t

| enum timer_compare_match_t | |
|---|---|
| Options for storing compare match value | |
| Enumerator | |
| TIMER_COMPARE_MATCH_A | Compare match A value. |
| TIMER_COMPARE_MATCH_B | Compare match B value. |
| TIMER_COMPARE_MATCH_C | Compare match C value. |
| TIMER_COMPARE_MATCH_D | Compare match D value. |
| TIMER_COMPARE_MATCH_E | Compare match E value. |
| TIMER_COMPARE_MATCH_F | Compare match F value. |
| TIMER_COMPARE_MATCH_G | Compare match G value. |
| TIMER_COMPARE_MATCH_H | Compare match H value. |

### ◆ timer_state_t

| enum timer_state_t | |
|---|---|
| Possible status values returned by timer_api_t::statusGet. | |
| Enumerator | |
| TIMER_STATE_STOPPED | Timer is stopped. |
| TIMER_STATE_COUNTING | Timer is running. |
| TIMER_STATE_UNKNOWN | Timer state could not be defined. |

### ◆ timer_mode_t

| enum timer_mode_t | |
|---|---|
| Timer operational modes | |
| Enumerator | |
| TIMER_MODE_8_BIT_COUNTER | 8-bit counter mode |
| TIMER_MODE_16_BIT_COUNTER | 16-bit counter mode |
| TIMER_MODE_32_BIT_COUNTER | 32-bit counter mode |
| TIMER_MODE_16_BIT_CAPTURE | 16-bit capture mode |
| TIMER_MODE_PERIODIC | Timer restarts after period elapses. |
| TIMER_MODE_ONE_SHOT | Timer stops after period elapses. |
| TIMER_MODE_PWM | Timer generates saw-wave PWM output. |
| TIMER_MODE_8_BIT_COUNTER | 8-bit counter mode |
| TIMER_MODE_16_BIT_COUNTER | 16-bit counter mode |
| TIMER_MODE_32_BIT_COUNTER | 32-bit counter mode |
| TIMER_MODE_16_BIT_CAPTURE | 16-bit capture mode |
| TIMER_MODE_PERIODIC | Timer restarts after period elapses. |
| TIMER_MODE_ONE_SHOT | Timer stops after period elapses. |
| TIMER_MODE_PWM | Timer generates saw-wave PWM output. |

| | |
|---|---|
| TIMER_MODE_8_BIT_COUNTER | 8-bit counter mode |
| TIMER_MODE_16_BIT_COUNTER | 16-bit counter mode |
| TIMER_MODE_32_BIT_COUNTER | 32-bit counter mode |
| TIMER_MODE_16_BIT_CAPTURE | 16-bit capture mode |
| TIMER_MODE_PERIODIC | Timer restarts after period elapses. |
| TIMER_MODE_ONE_SHOT | Timer stops after period elapses. |
| TIMER_MODE_PWM | Timer generates saw-wave PWM output. |
| TIMER_MODE_PERIODIC | Timer restarts after period elapses. |
| TIMER_MODE_ONE_SHOT | Timer stops after period elapses. |
| TIMER_MODE_PWM | Timer generates saw-wave PWM output. |
| TIMER_MODE_ONE_SHOT_PULSE | Saw-wave one-shot pulse mode (fixed buffer operation). |
| TIMER_MODE_TRIANGLE_WAVE_SYMMETRIC_PWM | Timer generates symmetric triangle-wave PWM output. |
| TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM | Timer generates asymmetric triangle-wave PWM output. |
| TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3 | Timer generates Asymmetric Triangle-wave PWM output. In PWM mode 3, the duty cycle does not need to be updated at each tough/crest interrupt. Instead, the trough and crest duty cycle values can be set once and only need to be updated when the application needs to change the duty cycle. |

### ◆ timer_direction_t

| enum timer_direction_t | |
|---|---|
| Direction of timer count | |
| Enumerator | |
| TIMER_DIRECTION_DOWN | Timer count goes up. |
| TIMER_DIRECTION_UP | Timer count goes down. |

### ◆ timer_source_div_t

| enum timer_source_div_t | |
|---|---|
| Clock source divisors | |
| Enumerator | |
| TIMER_SOURCE_DIV_1 | Timer clock source divided by 1. |
| TIMER_SOURCE_DIV_2 | Timer clock source divided by 2. |
| TIMER_SOURCE_DIV_4 | Timer clock source divided by 4. |
| TIMER_SOURCE_DIV_8 | Timer clock source divided by 8. |
| TIMER_SOURCE_DIV_16 | Timer clock source divided by 16. |
| TIMER_SOURCE_DIV_32 | Timer clock source divided by 32. |
| TIMER_SOURCE_DIV_64 | Timer clock source divided by 64. |
| TIMER_SOURCE_DIV_128 | Timer clock source divided by 128. |
| TIMER_SOURCE_DIV_256 | Timer clock source divided by 256. |
| TIMER_SOURCE_DIV_512 | Timer clock source divided by 512. |
| TIMER_SOURCE_DIV_1024 | Timer clock source divided by 1024. |
| TIMER_SOURCE_DIV_2048 | Timer clock source divided by 2048. |
| TIMER_SOURCE_DIV_4096 | Timer clock source divided by 4096. |
| TIMER_SOURCE_DIV_8192 | Timer clock source divided by 8912. |
| TIMER_SOURCE_DIV_16384 | Timer clock source divided by 16384. |
| TIMER_SOURCE_DIV_32768 | Timer clock source divided by 32768. |
| TIMER_SOURCE_DIV_1 | Timer clock source divided by 1. |
| TIMER_SOURCE_DIV_2 | Timer clock source divided by 2. |
| TIMER_SOURCE_DIV_4 | Timer clock source divided by 4. |
| TIMER_SOURCE_DIV_8 | Timer clock source divided by 8. |
| TIMER_SOURCE_DIV_16 | Timer clock source divided by 16. |

| | |
|---|---|
| TIMER_SOURCE_DIV_32 | Timer clock source divided by 32. |
| TIMER_SOURCE_DIV_64 | Timer clock source divided by 64. |
| TIMER_SOURCE_DIV_128 | Timer clock source divided by 128. |
| TIMER_SOURCE_DIV_256 | Timer clock source divided by 256. |
| TIMER_SOURCE_DIV_512 | Timer clock source divided by 512. |
| TIMER_SOURCE_DIV_1024 | Timer clock source divided by 1024. |
| TIMER_SOURCE_DIV_2048 | Timer clock source divided by 2048. |
| TIMER_SOURCE_DIV_4096 | Timer clock source divided by 4096. |
| TIMER_SOURCE_DIV_8192 | Timer clock source divided by 8912. |
| TIMER_SOURCE_DIV_16384 | Timer clock source divided by 16384. |
| TIMER_SOURCE_DIV_32768 | Timer clock source divided by 32768. |
| TIMER_SOURCE_DIV_1 | Timer clock source divided by 1. |
| TIMER_SOURCE_DIV_2 | Timer clock source divided by 2. |
| TIMER_SOURCE_DIV_4 | Timer clock source divided by 4. |
| TIMER_SOURCE_DIV_8 | Timer clock source divided by 8. |
| TIMER_SOURCE_DIV_16 | Timer clock source divided by 16. |
| TIMER_SOURCE_DIV_32 | Timer clock source divided by 32. |
| TIMER_SOURCE_DIV_64 | Timer clock source divided by 64. |
| TIMER_SOURCE_DIV_128 | Timer clock source divided by 128. |
| TIMER_SOURCE_DIV_256 | Timer clock source divided by 256. |
| TIMER_SOURCE_DIV_512 | Timer clock source divided by 512. |
| TIMER_SOURCE_DIV_1024 | Timer clock source divided by 1024. |
| TIMER_SOURCE_DIV_2048 | Timer clock source divided by 2048. |
| TIMER_SOURCE_DIV_4096 | Timer clock source divided by 4096. |

| | |
|---|---|
| TIMER_SOURCE_DIV_8192 | Timer clock source divided by 8912. |
| TIMER_SOURCE_DIV_16384 | Timer clock source divided by 16384. |
| TIMER_SOURCE_DIV_32768 | Timer clock source divided by 32768. |
| TIMER_SOURCE_DIV_1 | Timer clock source divided by 1. |
| TIMER_SOURCE_DIV_2 | Timer clock source divided by 2. |
| TIMER_SOURCE_DIV_4 | Timer clock source divided by 4. |
| TIMER_SOURCE_DIV_8 | Timer clock source divided by 8. |
| TIMER_SOURCE_DIV_16 | Timer clock source divided by 16. |
| TIMER_SOURCE_DIV_32 | Timer clock source divided by 32. |
| TIMER_SOURCE_DIV_64 | Timer clock source divided by 64. |
| TIMER_SOURCE_DIV_128 | Timer clock source divided by 128. |
| TIMER_SOURCE_DIV_256 | Timer clock source divided by 256. |
| TIMER_SOURCE_DIV_512 | Timer clock source divided by 512. |
| TIMER_SOURCE_DIV_1024 | Timer clock source divided by 1024. |
| TIMER_SOURCE_DIV_8192 | Timer clock source divided by 8192. |

## 5.3.18 Transfer

Interfaces

### Detailed Description

Transfer Interfaces.

### Modules

Transfer Interface

Interface for data transfer functions.

### 5.3.18.1 Transfer Interface

Interfaces » Transfer

#### Detailed Description

Interface for data transfer functions.

# Summary

The transfer interface supports background data transfer (no CPU intervention).

#### Data Structures

| | |
|---|---|
| struct | transfer_callback_args_t |
| struct | transfer_properties_t |
| struct | transfer_info_t |
| struct | transfer_cfg_t |
| struct | transfer_api_t |
| struct | transfer_instance_t |

#### Typedefs

| | |
|---|---|
| typedef void | transfer_ctrl_t |

#### Enumerations

| | |
|---|---|
| enum | transfer_mode_t |
| enum | transfer_size_t |
| enum | transfer_addr_mode_t |
| enum | transfer_repeat_area_t |
| enum | transfer_chain_mode_t |
| enum | transfer_irq_t |
| enum | transfer_start_mode_t |

#### Data Structure Documentation

### ◆ transfer_callback_args_t

| struct transfer_callback_args_t | | |
|---|---|---|
| Callback function parameter data. | | |
| Data Fields | | |
| void * | p_context | Placeholder for user data. Set in transfer_api_t::open function in transfer_cfg_t. |

### ◆ transfer_properties_t

| struct transfer_properties_t | | |
|---|---|---|
| Driver specific information. | | |
| Data Fields | | |
| uint32_t | block_count_max | Maximum number of blocks. |
| uint32_t | block_count_remaining | Number of blocks remaining. |
| uint32_t | transfer_length_max | Maximum number of transfers. |
| uint32_t | transfer_length_remaining | Number of transfers remaining. |

### ◆ transfer_info_t

| struct transfer_info_t | | |
|---|---|---|
| This structure specifies the properties of the transfer.<br><br>**Warning**<br>When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length.<br>When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t.<br>When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.<br>***Note***<br>*When using DTC, consider placing instances of this structure in a protected section of memory.* | | |
| Data Fields | | |
| union transfer_info_t | __unnamed__ | |
| void const *volatile | p_src | Source pointer. |
| void *volatile | p_dest | Destination pointer. |
| volatile uint16_t | num_blocks | Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC an DMAC) or TRANSFER_MODE_REPEAT (DMAC only) or TRANSFER_MODE_REPEAT_BLOCK (DMAC only), unused in other modes. |

| | | |
|---|---|---|
| volatile uint16_t | length | Length of each transfer. Range limited for TRANSFER_MODE_BLOCK, TRANSFER_MODE_REPEAT, and TRANSFER_MODE_REPEAT_BLOCK see HAL driver for details. |

### ◆ transfer_cfg_t

| struct transfer_cfg_t | | |
|---|---|---|
| Driver configuration set in transfer_api_t::open. All elements except p_extend are required and must be initialized. | | |
| Data Fields | | |
| transfer_info_t * | p_info | Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order. |
| void const * | p_extend | Extension parameter for hardware specific settings. |

### ◆ transfer_api_t

| struct transfer_api_t | |
|---|---|
| Transfer functions implemented at the HAL layer will follow this API. | |
| **Data Fields** | |
| fsp_err_t(* | open )(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg) |
| | |
| fsp_err_t(* | reconfigure )(transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info) |
| | |
| fsp_err_t(* | reset )(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers) |
| | |
| fsp_err_t(* | enable )(transfer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | disable )(transfer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | softwareStart )(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode) |
| | |
| fsp_err_t(* | softwareStop )(transfer_ctrl_t *const p_ctrl) |
| | |

| | |
|---|---|
| fsp_err_t(* | infoGet )(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties) |
| | |
| fsp_err_t(* | close )(transfer_ctrl_t *const p_ctrl) |
| | |
| fsp_err_t(* | reload )(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers) |
| | |
| fsp_err_t(* | callbackSet )(transfer_ctrl_t *const p_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory) |
| | |

## Field Documentation

### ◆ open

fsp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)

Initial configuration.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_cfg | Pointer to configuration structure. All elements of this structure must be set by user. |

### ◆ reconfigure

fsp_err_t(* transfer_api_t::reconfigure) (transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)

Reconfigure the transfer. Enable the transfer if p_info is valid.

**Parameters**

| | | |
|---|---|---|
| [in,out] | p_ctrl | Pointer to control block. Must be declared by user. Elements set here. |
| [in] | p_info | Pointer to a new transfer info structure. |

### ◆ reset

fsp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | p_src | Pointer to source. Set to NULL if source pointer should not change. |
| [in] | p_dest | Pointer to destination. Set to NULL if destination pointer should not change. |
| [in] | num_transfers | Transfer length in normal mode or number of blocks in block mode. In DMAC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC. |

### ◆ enable

fsp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::softwareStart is called if no peripheral event is chosen as activation source).

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ disable

fsp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)

Disable transfer. Transfers do not occur after the activation source event (or when transfer_api_t::softwareStart is called if no peripheral event is chosen as the DMAC activation source).

***Note***

*If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ softwareStart

fsp_err_t(* transfer_api_t::softwareStart) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

Start transfer in software.

**Warning**

Only works if no peripheral event is chosen as the DMAC activation source.

***Note***

*Not supported for DTC.*

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | mode | Select mode from transfer_start_mode_t. |

### ◆ softwareStop

fsp_err_t(* transfer_api_t::softwareStop) (transfer_ctrl_t *const p_ctrl)

Stop transfer in software. The transfer will stop after completion of the current transfer.

***Note***

*Not supported for DTC.*
*Only applies for transfers started with TRANSFER_START_MODE_REPEAT.*

**Warning**

Only works if no peripheral event is chosen as the DMAC activation source.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ infoGet

fsp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)

Provides information about this transfer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [out] | p_properties | Driver specific information. |

### ◆ close

fsp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)

Releases hardware lock. This allows a transfer to be reconfigured using transfer_api_t::open.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |

### ◆ reload

fsp_err_t(* transfer_api_t::reload) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const num_transfers)

To update next transfer information without interruption during transfer. Allow further transfer continuation.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | p_src | Pointer to source. Set to NULL if source pointer should not change. |
| [in] | p_dest | Pointer to destination. Set to NULL if destination pointer should not change. |
| [in] | num_transfers | Transfer length in normal mode or block mode. |

### ◆ callbackSet

| fsp_err_t(* transfer_api_t::callbackSet) (transfer_ctrl_t *const p_ctrl, void(*p_callback)(transfer_callback_args_t *), void *const p_context, transfer_callback_args_t *const p_callback_memory) |
|---|

Specify callback function and optional context pointer and working memory pointer.

**Parameters**

| | | |
|---|---|---|
| [in] | p_ctrl | Control block set in transfer_api_t::open call for this transfer. |
| [in] | p_callback | Callback function to register |
| [in] | p_context | Pointer to send to callback function |
| [in] | p_callback_memory | Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback. |

### ◆ transfer_instance_t

| struct transfer_instance_t | | |
|---|---|---|
| This structure encompasses everything that is needed to use an instance of this interface. | | |
| Data Fields | | |
| transfer_ctrl_t * | p_ctrl | Pointer to the control structure for this instance. |
| transfer_cfg_t const * | p_cfg | Pointer to the configuration structure for this instance. |
| transfer_api_t const * | p_api | Pointer to the API structure for this instance. |

## Typedef Documentation

### ◆ transfer_ctrl_t

| typedef void transfer_ctrl_t |
|---|
| Transfer control block. Allocate an instance specific control block to pass into the transfer API calls. |

## Enumeration Type Documentation

### ◆ transfer_mode_t

| enum transfer_mode_t | |
|---|---|
| Transfer mode describes what will happen when a transfer request occurs. | |
| Enumerator | |
| TRANSFER_MODE_NORMAL | In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers. |
| TRANSFER_MODE_REPEAT | Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers). |
| TRANSFER_MODE_BLOCK | In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers. |
| TRANSFER_MODE_REPEAT_BLOCK | In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.) |

### ◆ transfer_size_t

| enum transfer_size_t | |
|---|---|
| Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t | |
| Enumerator | |
| TRANSFER_SIZE_1_BYTE | Each transfer transfers a 8-bit value. |
| TRANSFER_SIZE_2_BYTE | Each transfer transfers a 16-bit value. |
| TRANSFER_SIZE_4_BYTE | Each transfer transfers a 32-bit value. |
| TRANSFER_SIZE_8_BYTE | Each transfer transfers a 64-bit value. |

### ◆ transfer_addr_mode_t

| enum transfer_addr_mode_t | |
|---|---|
| Address mode specifies whether to modify (increment or decrement) pointer after each transfer. | |
| Enumerator | |
| TRANSFER_ADDR_MODE_FIXED | Address pointer remains fixed after each transfer. |
| TRANSFER_ADDR_MODE_OFFSET | Offset is added to the address pointer after each transfer. |
| TRANSFER_ADDR_MODE_INCREMENTED | Address pointer is incremented by associated transfer_size_t after each transfer. |
| TRANSFER_ADDR_MODE_DECREMENTED | Address pointer is decremented by associated transfer_size_t after each transfer. |

### ◆ transfer_repeat_area_t

| enum transfer_repeat_area_t | |
|---|---|
| Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT_BLOCK, the selected pointer returns to its original value after each transfer. | |
| Enumerator | |
| TRANSFER_REPEAT_AREA_DESTINATION | Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK. |
| TRANSFER_REPEAT_AREA_SOURCE | Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK. |

### ◆ transfer_chain_mode_t

| enum transfer_chain_mode_t | |
|---|---|
| Chain transfer mode options.<br><br>***Note***<br>*Only applies for DTC.* | |
| Enumerator | |
| TRANSFER_CHAIN_MODE_DISABLED | Chain mode not used. |
| TRANSFER_CHAIN_MODE_EACH | Switch to next transfer after a single transfer from this transfer_info_t. |
| TRANSFER_CHAIN_MODE_END | Complete the entire transfer defined in this transfer_info_t before chaining to next transfer. |

### ◆ transfer_irq_t

| enum transfer_irq_t | |
|---|---|
| Interrupt options. | |
| Enumerator | |
| TRANSFER_IRQ_END | Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete.<br><br>Warning<br>DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete. |
| TRANSFER_IRQ_EACH | Interrupt occurs after each transfer.<br><br>*Note*<br>*Not available in all HAL drivers. See HAL driver for details.* |

### ◆ transfer_start_mode_t

| enum transfer_start_mode_t | |
|---|---|
| Select whether to start single or repeated transfer with software start. | |
| Enumerator | |
| TRANSFER_START_MODE_SINGLE | Software start triggers single transfer. |
| TRANSFER_START_MODE_REPEAT | Software start transfer continues until transfer is complete. |

# Chapter 6 Copyright

Copyright [2020-2025] Renesas Electronics Corporation and/or its affiliates. All Rights Reserved.

This software and documentation are supplied by Renesas Electronics America Inc. and may only be used with products of Renesas Electronics Corp. and its affiliates ("Renesas"). No other uses are authorized. Renesas products are sold pursuant to Renesas terms and conditions of sale. Purchasers are solely responsible for the selection and use of Renesas products and Renesas assumes no liability. No license, express or implied, to any intellectual property right is granted by Renesas. This software is protected under all applicable laws, including copyright laws. Renesas reserves the right to change or discontinue this software and/or this documentation. THE SOFTWARE AND DOCUMENTATION IS DELIVERED TO YOU "AS IS," AND RENESAS MAKES NO REPRESENTATIONS OR WARRANTIES, AND TO THE FULLEST EXTENT PERMISSIBLE UNDER APPLICABLE LAW, DISCLAIMS ALL WARRANTIES, WHETHER EXPLICITLY OR IMPLICITLY, INCLUDING WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT, WITH RESPECT TO THE SOFTWARE OR DOCUMENTATION. RENESAS SHALL HAVE NO LIABILITY ARISING OUT OF ANY SECURITY VULNERABILITY OR BREACH. TO THE MAXIMUM EXTENT PERMITTED BY LAW, IN NO EVENT WILL RENESAS BE LIABLE TO YOU IN CONNECTION WITH THE SOFTWARE OR DOCUMENTATION (OR ANY PERSON OR ENTITY CLAIMING RIGHTS DERIVED FROM YOU) FOR ANY LOSS, DAMAGES, OR CLAIMS WHATSOEVER, INCLUDING, WITHOUT LIMITATION, ANY DIRECT, CONSEQUENTIAL, SPECIAL, INDIRECT, PUNITIVE, OR INCIDENTAL DAMAGES; ANY LOST PROFITS, OTHER ECONOMIC DAMAGE, PROPERTY DAMAGE, OR PERSONAL INJURY; AND EVEN IF RENESAS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS, DAMAGES, CLAIMS OR COSTS.

Renesas FSP
Copyright © (2025) Renesas Electronics Corporation. All Rights Reserved.

User’s Manual

Publication Date: Revision 5.20 Oct.15.2025

# Renesas FSP V6.2.0

# User’s Manual

Renesas Electronics Corporation